TOKENISATION OVER BOUNDED ALPHABETS IS HARD

Anonymous authors
Paper under double-blind review

ABSTRACT

Recent works have proven tokenisation to be NP-complete. However, their proofs' constructions rely on tokenisation being applied to inputs with alphabets of unbounded cardinality, which does not accurately reflect the real world. Indeed, since practical applications of tokenisers involve fixed-size alphabets (e.g., Unicode or bytes), the implications of such a statement may be challenged. In this work, we examine the computational complexity of tokenisation over bounded alphabets, considering two variants of this problem: **bottom-up tokenisation** and **direct tokenisation**, where we must, respectively, select a sequence of merge operations (in bottom-up tokenisation) or a vocabulary (in direct tokenisation) whose application compresses a dataset to at most δ symbols. When alphabets are bounded to have only 2 characters, we do not only prove that bottom-up and direct tokenisation are NP-complete, but also that there is no polynomial-time approximation scheme for either of these problems (unless $\mathsf{P} = \mathsf{NP}$). Furthermore, even when alphabets are bounded to contain a single character, we can still prove the NP-completeness of direct tokenisation. Although the single-character case is not practical on its own, proving hardness results for an n-ary alphabet allows us to prove the same results for alphabets of any larger size. We thus conclude that direct tokenisation over any alphabet is NP-complete, and that both bottom-up and direct tokenisation do not admit polynomial-time approximation schemes for any alphabet of size 2 or larger.

1 INTRODUCTION

Tokenisation is the first step in most natural language processing pipelines. Given a character string $\mathbf{c}$, a tokeniser maps it to a sequence of subwords $\mathbf{s}$. Language models then operate on these subword sequences rather than the raw characters. Despite its central role, however, we still lack a comprehensive understanding of tokenisation; e.g., which properties of the produced subwords $\mathbf{s}$ actually help downstream modelling? A common property to aim for is **compression** (Sennrich et al., 2016; Uzan et al., 2024; Zouhar et al., 2023b), as using shorter subword-strings to encode a dataset allows for more efficient training and inference—more data can be passed through the model with the same number of flops. While not a silver bullet (Schmidt et al., 2024; Ali et al., 2024), compression has been shown to correlate with downstream model performance (Gallé, 2019; Rust et al., 2021; Zouhar et al., 2023a; Goldman et al., 2024) and will be our work's focus.

A practical concern follows immediately: once an objective (e.g., compression) is fixed, can an optimal tokeniser be found efficiently? Popular algorithms such as BPE and UnigramLM are greedy or heuristic and need not return an optimal tokeniser for their stated criteria. Further, recent work has sharpened this picture, proving the NP-completeness of finding an optimal tokeniser under a compression-style objective (Kozma and Voderholzer, 2024; Whittington et al., 2025; Lim et al., 2025). These papers, however, show this for the tokenisation of strings over unboundedly large alphabets. Conversely, in practice the strings we care about are typically composed of Unicode characters or bytes, thus using bounded alphabets. Whether it is possible to efficiently find optimal tokenisers over Unicode-strings (which have an alphabet size of roughly 150,000), byte-strings (with an alphabet size of 256), or bit-strings (with an alphabet size of 2) are open questions of practical relevance.

In this paper, we first define the **n-ary tokenisation problem**: the problem of finding an optimal tokeniser on strings constrained to alphabets of size n. We examine this problem under two variants: direct and bottom-up tokenisation, where—given a dataset over an n-ary alphabet and a vocabulary size K—we must find the vocabulary (in direct tokenisation) or sequence of merges (in bottom-up tokenisation) which when applied to the dataset maximally compresses it. We prove that (i) both

direct and bottom-up binary tokenisation do not admit a polynomial-time approximation scheme (ptas;[1] unless P = NP),[2] and (ii) that for the direct case, even unary tokenisation is NP-complete. Notably, unary and binary are the easiest of the n-ary tokenisation problems, and thus these hardness results also trivially extend to tokenisation problems with larger alphabets.

Our results thus indicate that the computational hardness of tokenisation is not an artifact of large alphabets or elaborate merge operations: it already appears under direct tokenisation over unary alphabets. This helps explain why practical algorithms (e.g., BPE) rely on approximations, and suggests that future work should focus on provably good approximate methods or on relaxations for this problem.

2 TOKENISATION

We follow Whittington et al.'s 2025 notation and colour-coding

- Blue for raw data (i.e., characters $\mathbf{c} \in \Sigma^*$);
- Magenta for tokeniser-specific data (i.e., subwords $\mathbf{s} \in \mathcal{S}^*$ and merges $\mathbf{m} \in \mathcal{M}^*$);
- Orange for functions (e.g., $\texttt{tok}$).

Let $\mathbf{c} \in \Sigma^*$ be a **character-string**, composed of characters c from an alphabet Σ; for notational convenience, we may write one such string as $\mathbf{c} = c_1 c_2 \ldots c_{|\mathbf{c}|}$. Character-strings compose the raw text data found, say, on the web, which make up the datasets on which language models are trained. We denote one such dataset by $\mathcal{D} = \{\mathbf{c}_m\}_{m=1}^{M}$. Before feeding data to our models, however, we typically convert them to strings of subwords, which is the job of a tokeniser.

Formally, a tokeniser can be defined as a tuple $\langle \mathcal{S}, \texttt{detok}, \texttt{tok} \rangle$, composed of a vocabulary, a decoding and an encoding function. A **vocabulary** is a finite set of subwords, each of which is a non-empty span of characters; we thus write $\mathcal{S} \subset \Sigma^+$. A **subword-string** is then a sequence $\mathbf{s} \in \mathcal{S}^*$ and represents a character-string via the concatenation of its subwords' characters: we say that a pair of character- and subword-strings are equivalent if

$$\mathbf{c} \overset{\circ}{=} \mathbf{s} \iff \mathbf{c} = \texttt{concat}(\mathbf{s}), \qquad \texttt{concat}(\mathbf{s}) = s_1 \circ s_2 \circ \cdots \circ s_{|\mathbf{s}|} \tag{1}$$

where we write $\mathbf{s} = \langle s_1, s_2, \cdots, s_{|\mathbf{s}|} \rangle$ and each $s_t \in \mathcal{S}$ is a subword. Notably, $\Sigma \subseteq \mathcal{S}$ is typically enforced to guarantee that every $\mathbf{c} \in \Sigma^*$ can be represented by at least one subword-string $\mathbf{s} \in \mathcal{S}^*$, and we say that a vocabulary's size is $|\mathcal{S}| = |\Sigma| + K$. Second in this tuple, a **decoding function** is defined as $\texttt{detok}\colon \mathcal{S}^* \to \Sigma^*$, and given a subword-string it outputs the character-string it represents. This function thus is simply defined as $\texttt{detok}(\mathbf{s}) \overset{\text{def}}{=} \texttt{concat}(\mathbf{s})$.

Finally, an **encoding function** $\texttt{tok}\colon \Sigma^* \to \mathcal{S}^*$ maps character- to subword-strings while ensuring the equivalence $\mathbf{c} \overset{\circ}{=} \mathbf{s}$ for $\mathbf{s} = \texttt{tok}(\mathbf{c})$. Several encoding functions may respect this constraint, as many subword-strings may be equivalent to a character-string. For instance, given $\mathcal{S} = \{a, aa, aaa\}$, the string $\mathbf{c} = aaa$ could be tokenised as $\mathbf{s} = \langle aaa \rangle$ or as $\mathbf{s} = \langle a, aa \rangle$. We focus on two encoding functions in this paper, which we follow Whittington et al. (2025) in labelling as direct and bottom-up. The **direct encoding function** ($\texttt{tok}_{\Diamond}$) only requires a vocabulary, which it applies optimally to encode a character-string. In turn, the **bottom-up encoding function** ($\texttt{tok}_{\uparrow}$) takes a merge sequence $\mathbf{m} = \langle m_1, \ldots, m_K \rangle$ as input, which it applies in order to a character-string; each of these merges is composed of a pair of subwords, $m_k = (s_k^{[1]}, s_k^{[2]})$, and we write $\texttt{merge}_m\colon \mathcal{S}^* \to \mathcal{S}^*$ to represent a function which, given a subword-string, processes it left-to-right and replaces any consecutive occurrence of the pair $s_k^{[1]}, s_k^{[2]}$ with a new token $s_k^{[\text{new}]} = s_k^{[1]} \circ s_k^{[2]}$. We formalise these encoding functions as

$$\texttt{tok}_{\Diamond}[\mathcal{S}](\mathbf{c}) \overset{\text{def}}{=} \underset{\substack{\mathbf{s} \in \mathcal{S}^* \\ \text{s.t. } \mathbf{c} \overset{\circ}{=} \mathbf{s}}}{\arg\min} |\mathbf{s}|, \qquad \texttt{tok}_{\uparrow}[\mathbf{m}](\mathbf{c}) \overset{\text{def}}{=} \left(\bigodot_{m \in \mathbf{m}} \texttt{merge}_m \right)(\mathbf{c}) \tag{2}$$

where $\bigodot$ represents function composition. A tokeniser is thus fully determined by a vocabulary or merge-sequence; for the direct case we have $\texttt{tok} \overset{\text{def}}{=} \texttt{tok}_{\Diamond}[\mathcal{S}]$ while for bottom-up $\texttt{tok} \overset{\text{def}}{=} \texttt{tok}_{\uparrow}[\mathbf{m}]$. Importantly, as shown by Schmidt et al. (2024), the direct encoding function ($\texttt{tok}_{\Diamond}[\mathcal{S}]$) can be efficiently computed in $O(|\mathbf{c}|^2)$ time.

[1] We use the lower-case ptas acronym to denote the corresponding class of polynomial-time approximation algorithms, and the upper-case PTAS to refer to the complexity class of problems for which a ptas exists.

[2] More specifically, we present a constant lower bound on the approximation ratio achievable by any polynomial-time algorithm on this problem.

2.1 Objective Functions and their Optimisation

As described above, a direct tokeniser is fully determined by a vocabulary, while a bottom-up tokeniser is identified by a merge-sequence. How to select a specific tokeniser, though? This is typically done via defining an objective function $\mathfrak{G}$ which, given an encoding function ($\texttt{tok}$) and a dataset ($\mathcal{D}$), returns a value representing the cost of that particular choice. Choosing a tokeniser then "simply" requires optimising this objective: e.g., for direct tokenisation we must find $\mathcal{S}_{\texttt{opt}} = \arg\min_{\mathcal{S} \subset \Sigma^+} \mathfrak{G}(\texttt{tok}_{\mathcal{S}}[\mathcal{S}], \mathcal{D})$ under the constraint that $|\mathcal{S}| = |\Sigma| + K$.

Several objective functions exist. UnigramLM (Kudo, 2018), for instance, selects a vocabulary which optimises a dataset's unigram negative log-probability. Other work has proposed alternative measures, such as the frequency of the 5-th % least frequent token (Gowda and May, 2020), or the tokeniser's Rényi efficiency (Zouhar et al., 2023a). As mentioned above, we focus on compression in this paper. We do so following a battery of previous work which formally analyses tokenisers (Zouhar et al., 2023b; Kozma and Voderholzer, 2024; Whittington et al., 2025; Lim et al., 2025). Prior work has shown that a tokeniser's compression correlates with the downstream performance of language models trained on its output subword-strings (Gallé, 2019; Zouhar et al., 2023a). We note, however, that other recent work has criticised compression as the sole objective for tokenisation, showing that these two properties (compression and downstream performance) may have a more complex relationship than originally suspected (Ali et al., 2024; Schmidt et al., 2024).

There are two natural ways to define a compression objective: **compressed length**, which measures the number of remaining symbols after a string is tokenised, and **compression reduction**, which measures how many symbols are reduced in the string by a tokeniser. These are formalised as:

$$\underbrace{\mathfrak{G}_\ell(\texttt{tok}, \mathcal{D}) \stackrel{\text{def}}{=} \sum_{\mathbf{c} \in \mathcal{D}} |\texttt{tok}(\mathbf{c})|}_{\textbf{compressed length}\text{, size of remaining string}}, \qquad \underbrace{\mathfrak{G}_\mathsf{r}(\texttt{tok}, \mathcal{D}) \stackrel{\text{def}}{=} \sum_{\mathbf{c} \in \mathcal{D}} \Big(|\texttt{tok}(\mathbf{c})| - |\mathbf{c}|\Big)}_{\textbf{compression reduction}\text{, number of reduced symbols}} \tag{3}$$

While equivalent in how they rank tokenisers, this choice can make a big difference when evaluating the quality of an approximation. When using minimisation objectives, such as $\mathfrak{G}_\ell$, the **approximation ratio** of an algorithm upper-bounds the ratio between the objective value achieved by the algorithm's solution and an optimal solution, being thus at least 1 by definition. A similar definition applies when using maximisation objectives, such as $\mathfrak{G}_\mathsf{r}$, but the approximation ratio is inversed. We say we have a **δ-approximation algorithm** if, for every possible input, this ratio is bound from above by δ. If a dataset has 1,000 characters and would have 100 symbols if optimally compressed, a suboptimal tokeniser which instead reduces it to at most 200 symbols would have an approximation ratio of 2 under $\mathfrak{G}_\ell$ but of 1.125 under $\mathfrak{G}_\mathsf{r}$. Notably, prior work has analysed both these measures. We argue here that compressed length is a more natural objective, as it directly relates to the throughput achieved by a language model processing that text, being thus connected to the model's training and inference costs. A 2-approximation for $\mathfrak{G}_\ell$ implies that a language model using that tokeniser may be 2-times slower (and more costly) than optimal when processing the same text.[3]

After deciding on an objective function, such as $\mathfrak{G}_\ell$ above, we must select a vocabulary ($\mathcal{S} \subset \Sigma^+$) or merge-sequence ($\mathbf{m} \in \mathcal{M}^*$) which optimises it. Unfortunately, both these optimisation problems have infinite search spaces (respectively, $\mathcal{P}(\Sigma^+)$ and $\mathcal{M}^*$, where $\mathcal{P}$ denotes the powerset operation), which begs the question: is there an efficient way to find these optima? Recent work has shown that, in general, this is not possible, proving compression-based tokenisation to be NP-complete; namely, Kozma and Voderholzer (2024) showed this for bottom-up tokenisation, Whittington et al. (2025) for direct and bottom-up tokenisation, and Lim et al. (2025) for direct tokenisation with candidate tokens. This means that, unless P = NP , there exists no polynomial-time algorithm to find compression-optimal tokenisers. Beyond that, using the $\mathfrak{G}_\mathsf{r}$ objective function, Kozma and Voderholzer (2024) showed that bottom-up tokenisation is not only NP-complete, but that it is not in the **polynomial-time approximation scheme** (PTAS) complexity class (unless P = NP). The PTAS class is characterised by problems for which a $\texttt{ptas}$ exists: for every constant $\varepsilon > 0$, there exists a polynomial-time algorithm (whose run-time may depend on ε), which solves it with an approximation ratio upper-bounded by $1+\varepsilon$. Not being in PTAS thus implies that there is no polynomial-time algorithm which can approximate the optimal solution with approximation ratios arbitrarily close to 1. Notably, all of these complexity proofs rely on tokenisation problems over alphabets of unbounded size. Whether these results hold once alphabet sizes are bounded is thus left open.

[3] Assuming that language models cannot achieve sub-linear computational complexity on their input's length.

3 TOKENISATION OVER BOUNDED ALPHABETS

We now move to the analysis of tokenisation over bounded alphabets. Let an ***n*-ary alphabet** be an alphabet with size $|\Sigma| = n$. We define the tokenisation problem over such bounded alphabets as follows.

Definition 1. *Let K be a vocabulary size and $\mathcal{D}$ be a dataset composed of character-strings from an alphabet of size $|\Sigma| = n$. For a given δ, the **n-ary tokenisation decision problem** requires deciding whether there exists a vocabulary $\mathcal{S}_{\texttt{opt}} \subseteq \Sigma^+$ (for direct tokenisation) or a merge-sequence $\mathbf{m}_{\texttt{opt}} \in \mathcal{M}^*$ (for bottom-up tokenisation) which compresses $\mathcal{D}$ to at most δ symbols. The **n-ary tokenisation optimisation problem** is to find what the maximal such compression of $\mathcal{D}$ is. Formally:*

$$\underbrace{\delta \geq \min_{\texttt{tok}\in\mathcal{T}} \sum_{\mathbf{c}\in\mathcal{D}} |\texttt{tok}(\mathbf{c})|, \text{ s.t. } |\texttt{tok}| = K}_{n\text{-ary tokenisation decision problem}}, \quad \underbrace{\delta_{\texttt{opt}} = \min_{\texttt{tok}\in\mathcal{T}} \sum_{\mathbf{c}\in\mathcal{D}} |\texttt{tok}(\mathbf{c})|, \text{ s.t. } |\texttt{tok}| = K}_{n\text{-ary tokenisation optimisation problem}} \quad (4)$$

where $\mathcal{T} \stackrel{\text{def}}{=} \{\texttt{tok}_{\Diamond}[\mathcal{S}] \mid \mathcal{S} \subset \Sigma^+\}$ for direct tokenisation and $\mathcal{T} \stackrel{\text{def}}{=} \{\texttt{tok}_{\uparrow}[\mathbf{m}] \mid \mathbf{m} \in \mathcal{M}^*\}$ for bottom-up.

We will more specifically call these the direct n-ary tokenisation problem and the bottom-up n-ary tokenisation problem when dealing with, respectively, direct and bottom-up tokenisers, writing $\text{Tok}^n_{\Diamond}(\mathcal{D}, K, \delta)$ and $\text{Tok}^n_{\uparrow}(\mathcal{D}, K, \delta)$ for the functions which return the solution to their decision problems. Notably, the n-ary tokenisation problems form a clear hierarchy from easiest ($n = 1$) to hardest ($n \to \infty$), with unary tokenisation being the easiest such problem. In the next sections, we first prove that both direct and bottom-up binary tokenisation are hard to approximate, i.e., that both these problems are not in PTAS (in §4). We then prove that direct binary tokenisation is NP-complete (in §5).

Fact 1. *If n-ary tokenisation is NP-hard, all n'-ary tokenisation problems for $n' > n$ are NP-hard.*

Proof. Let $n, n' \in \mathbb{N}$ with $n' \geq n$. Any instance of the n-ary tokenisation problem is a valid instance of the n'-ary problem with the same solutions, allowing for a trivial reduction between them. Thus, any proof of hardness for the n-ary tokenisation problem immediately applies to n'-ary problems. □

A Note on Optimisation vs. Decision Problems. Typically, NP-hardness is defined as a property of decision problems, while hardness of approximation (and consequently, being contained or not in PTAS) is a notion regarding optimisation problems. There is, however, a notion of equivalence between these classes of problems: if a polynomial-time algorithm exists to solve a decision problem (i.e., if this problem is not NP-hard), it can usually be leveraged to also find an efficient algorithm for its associated optimisation problem, and *vice-versa*. Similarly, if no polynomial-time algorithm can solve an optimisation problem with an approximation ratio arbitrarily close to 1 (i.e., if the problem is not in PTAS), this implies that there must be some constant ε such that it is NP-hard to distinguish between instances that admit a solution of quality x and those that admit a solution of quality $(1 + \varepsilon)x$. We will use this latter property here to show hardness of approximation, relying on gap-preserving reductions.[4] To this end, it will be useful to also define gap versions of the problems we discuss. Formally, we will denote such gap versions similarly to their decision versions (e.g., $\text{Tok}^n_{\Diamond}(\mathcal{D}, K, \delta)$ above), but while providing two decision boundaries instead (e.g., $\text{Tok}^n_{\Diamond}(\mathcal{D}, K, (\delta^-, \delta^+))$). In minimisation gap problems, the task is then to decide whether their optimal value is at most δ^+ or at least δ^- (with the opposite being true for maximisation problems); if a value falls between these, any answer is acceptable. For the n-ary tokenisation problems, for instance, we would require an algorithm which computes:

$$\text{Tok}^n(\mathcal{D}, K, (\delta^-, \delta^+)) = \begin{cases} \texttt{T} & \texttt{if } \delta^+ \geq \min_{\texttt{tok}\in\mathcal{T}} \sum_{\mathbf{c}\in\mathcal{D}} |\texttt{tok}(\mathbf{c})|, \text{ s.t. } |\texttt{tok}| = K \\ \texttt{F} & \texttt{elif } \delta^- \leq \min_{\texttt{tok}\in\mathcal{T}} \sum_{\mathbf{c}\in\mathcal{D}} |\texttt{tok}(\mathbf{c})|, \text{ s.t. } |\texttt{tok}| = K \\ ? & \texttt{else} \end{cases} \quad (5)$$

4 BINARY TOKENISATION IS HARD TO DECIDE AND APPROXIMATE

In this section, we will prove NP-hardness of the two binary tokenisation decision problems above, and of their corresponding gap problems (for specific gaps). To this end, we will use a reduction from the **3-occurrence maximum 2-satisfiability** problem (3-OCC-MAX2SAT), which we define in §4.1. We then move on to proving results showing hardness of approximation for the direct and bottom-up binary tokenisation problems (in §4.2 and §4.3, respectively).

[4] We note that hardness of approximation is not formally the same as proving APX-hardness (as was done in Kozma and Voderholzer, 2024). However, it allows for the same conclusion: a PTAS for the binary (and larger) tokenisation problems cannot exist, unless P = NP. Additionally, our gap-preserving reductions allow us to find explicit constants to which the problems cannot be approximated in polynomial time, again unless P = NP.

4.1 3-Occurrence Maximum 2-Satisfiability

Let X be a Boolean variable which is assigned a value $x \in \{\texttt{F}, \texttt{T}\}$, and $\mathcal{X} = \{X_j\}_{j=1}^{J}$ be a set of such variables, with joint assignment $\chi = \{x_j\}_{j=1}^{J}$. Further, let $\mathcal{C} = \{(L_i^1 \vee L_i^2)\}_{i=1}^{I}$ be a set of clauses,[5] where each literal L is either a variable X_j or its negation $\neg X_j$. We define 3-OCC-MAX2SAT as follows.

Definition 2. *Let $\mathcal{X} = \{X_j\}_{j=1}^{J}$ be a set of Boolean variables and $\mathcal{C} = \{(L_i^1 \vee L_i^2)\}_{i=1}^{I}$ be a set of clauses. Further, let each variable X_j occur in exactly three clauses. Given a target $\gamma \in \mathbb{N}$, the* **3-OCC-MAX2SAT** ***decision problem*** *requires deciding whether there exists an assignment $\chi \in \{\texttt{F}, \texttt{T}\}^J$ such that at least γ clauses are satisfied. The* **3-OCC-MAX2SAT** ***optimisation problem*** *requires finding the maximum number of satisfiable clauses. Formally:*

$$\underbrace{\gamma \leq \max_{\chi \in \{\texttt{F},\texttt{T}\}^J} \sum_{i=1}^{I} \mathbb{1}_\chi\{L_i^1 \vee L_i^2\}}_{\texttt{3-OCC-MAX2SAT } \textit{decision problem}} \qquad \underbrace{\gamma_{\texttt{opt}} = \max_{\chi \in \{\texttt{F},\texttt{T}\}^J} \sum_{i=1}^{I} \mathbb{1}_\chi\{L_i^1 \vee L_i^2\},}_{\texttt{3-OCC-MAX2SAT } \textit{optimisation problem}} \tag{6}$$

We will write $\texttt{3OM2S}(\mathcal{X}, \mathcal{C}, \gamma)$ to denote a function which, given a certain instance of the 3-OCC-MAX2SAT decision problem, returns its solution. The 3-OCC-MAX2SAT problem was proven to be hard to approximate by Berman and Karpinski (1999). As not belonging to PTAS implies NP-hardness, this problem is also NP-hard.

4.2 Direct Binary Tokenisation is Hard to Decide and Approximate

In this section, we prove that the direct binary tokenisation problem is both hard to decide and to approximate beyond a certain constant $r > 1$. First, we will prove that the *decision* version is NP-hard (in §4.2.1). Second, we will then use this initial result to prove that a *gap* version of the problem is similarly NP-hard (in §4.2.2). This will complete our proof that this problem's *optimisation* version is hard to approximate, as being contained in PTAS would allow us to solve the gap problem.

4.2.1 The Direct Binary Tokenisation Decision Problem is NP-Hard

We now prove NP-completeness of direct binary tokenisation, which requires two things: inclusion in NP and being NP-hard. Inclusion in NP follows from the general (unbounded) case, which was previously proven by Whittington et al. (2025). Proving NP-hardness requires a polynomial-time reduction from another NP-hard problem to this problem, which we will design in what follows.

Reduction 1. *Consider an instance of the* 3-OCC-MAX2SAT *decision problem and a binary alphabet $\Sigma = \{0, 1\}$. Now, for each variable X_j, let $\mathbf{x}_j^{\texttt{T}} = 0^{2j-1}$ and $\mathbf{x}_j^{\texttt{F}} = 0^{2j}$, i.e., character-strings formed of 0 repeated $2j-1$ or $2j$ times. Then we build subdatasets:*

$$\mathcal{D}_1 = \{1\mathbf{x}_j^{\texttt{T}},\ \mathbf{x}_j^{\texttt{T}}1,\ 1\mathbf{x}_j^{\texttt{F}},\ \mathbf{x}_j^{\texttt{F}}1 \mid 1 \leq j \leq J\} \times f, \qquad \mathcal{D}_2 = \{1\mathbf{x}_j^{\texttt{T}}1,\ 1\mathbf{x}_j^{\texttt{F}}1 \mid 1 \leq j \leq J\} \times f' \tag{7a}$$

$$\mathcal{D}_3 = \{1\mathbf{x}_j^{\texttt{T}}1\mathbf{x}_j^{\texttt{F}}1 \mid 1 \leq j \leq J\} \times f'', \qquad \mathcal{D}_4 = \{1L_i^1 1 L_i^2 1 \mid 1 \leq i \leq I\} \times 1 \tag{7b}$$

where $\times f$ denotes that a set of strings should be repeated f times in the corresponding dataset. These multiplicities are $f'' \stackrel{\text{def}}{=} 7$, $f' \stackrel{\text{def}}{=} 2(f''+3)+1 = 21$, $f \stackrel{\text{def}}{=} 2(f'+f''+3)+1 = 63$. A full dataset is then formed by joining these subdatasets: $\mathcal{D} = \mathcal{D}_1 \cup \mathcal{D}_2 \cup \mathcal{D}_3 \cup \mathcal{D}_4$. Finally, we set the number of allowed tokens $K = 5J$ and the target compression $\delta = 4fJ + 3f'J + 2f''J + 3I - \gamma = 329J + 3I - \gamma$.[6]

We will write $\texttt{R1}(\mathcal{X}, \mathcal{C}, \gamma)$ to represent the D-2-TOK instance which is output by this reduction, represented by the tuple $(\mathcal{D}, K, \delta)$. Notably, this reduction runs in polynomial time. By proving its correctness, thus, we can show that direct binary tokenisation is an NP-hard problem. For this reduction to be correct, a 3-OCC-MAX2SAT instance must be satisfiable if and only if its reduced tokenisation instance is as well, i.e.,: $\texttt{3OM2S}(\mathcal{X}, \mathcal{C}, \gamma) \iff \texttt{Tok}_\diamond^2(\texttt{R1}(\mathcal{X}, \mathcal{C}, \gamma))$. We now set out to prove both directions of this iff clause.

Theorem 1. *The direct binary tokenisation decision problem is* NP*-complete.*

Proof sketch. This proof is done in two steps.

[5] In some formalisations, 3-OCC-MAX2SAT allows clauses of size one. We work here, more specifically, with the 3-occurrence maximum exact-2-satisfiability variant of this problem, thus not allowing single literal clauses.

[6] This reduction is inspired by Whittington et al.'s 2025 reduction, which we update to (i) rely on binary, as opposed to unbounded, alphabets; (ii) use constant-sized f's, which allow us to prove approximation hardness.

Forward step. ($\texttt{3OM2S}(\mathcal{X},\mathcal{C},\gamma) \implies \texttt{Tok}^2_\$(\texttt{R1}(\mathcal{X},\mathcal{C},\gamma))$) See a formal proof in Lemma 1 in App. A. Assuming an instance of `3-OCC-MAX2SAT` is satisfied by assignment $\mathcal{X}^\star = \{x_j^\star\}_{j=1}^J$, we build a direct tokeniser with tokens $1\mathtt{x}_j^\mathtt{T}$, $\mathtt{x}_j^\mathtt{T}1$, $1\mathtt{x}_j^\mathtt{F}$, $\mathtt{x}_j^\mathtt{F}1$, and with token $1\mathtt{x}_j^\mathtt{T}1$ if $x_j^\star = \mathtt{T}$ else $1\mathtt{x}_j^\mathtt{F}1$. This tokeniser compresses $\mathcal{D}_1$ to $252J$, $\mathcal{D}_2$ to $63J$, $\mathcal{D}_3$ to $14J$, and $\mathcal{D}_4$ to $3I - \gamma^\star$ tokens, where $\gamma^\star$ is the number of clauses satisfied by $\mathcal{X}^\star$. Adding these compressed lengths together, we find that they satisfy the direct tokenisation problem, as $\gamma^\star \geq \gamma$ by assumption.

Backward step. ($\texttt{Tok}^2_\$(\texttt{R1}(\mathcal{X},\mathcal{C},\gamma)) \implies \texttt{3OM2S}(\mathcal{X},\mathcal{C},\gamma)$) See full proof in Lemma 2 in App. B. We first show that an optimal tokeniser for the `D-2-TOK` instance is always **`sat`-compliant**: it contains all tokens of the form $1\mathtt{x}_j^\mathtt{T}, \mathtt{x}_j^\mathtt{T}1, 1\mathtt{x}_j^\mathtt{F}, \mathtt{x}_j^\mathtt{F}1$, and either $1\mathtt{x}_j^\mathtt{T}1$ or $1\mathtt{x}_j^\mathtt{F}1$ for each $j \in \{1,\ldots,J\}$. We do this by showing that $\mathcal{D}_1$ guarantees that any optimal solution includes tokens $1\mathtt{x}_j^\mathtt{T}, \mathtt{x}_j^\mathtt{T}1, 1\mathtt{x}_j^\mathtt{F}, \mathtt{x}_j^\mathtt{F}1$; $\mathcal{D}_2$ guarantees that any optimal solutions further only include tokens of the form $1\mathtt{x}_j^\mathtt{T}1, 1\mathtt{x}_j^\mathtt{F}1$; and $\mathcal{D}_3$ guarantees that either token $1\mathtt{x}_j^\mathtt{T}1$ or $1\mathtt{x}_j^\mathtt{F}1$ exist for each $j \in \{1,\ldots,J\}$. Then, we show that if such a `sat`-compliant tokeniser reaches the desired compression, it must correspond to an assignment $\mathcal{X}^\star$ which satisfies the desired number of clauses. □

4.2.2 The Direct Binary Tokenisation Gap Problem is NP-hard

We now prove that not only the decision version of the direct binary tokenisation problem is NP-hard, but so is its gap version. Proving NP-hardness of a gap problem is an indirect way of proving that its optimisation version is hard to approximate: if an efficient algorithm can approximate the optimisation problem arbitrarily well (which is thus contained in PTAS), it could be used to solve the gap problem.

Theorem 2. *The direct binary tokenisation gap problem is* NP*-hard. Thus, the direct binary tokenisation optimisation problem is not in* PTAS*, unless* P = NP.

Proof sketch. See a formal proof in Lemma 3 in App. C. As shown by Berman and Karpinski (1998; 1999), the `3-OCC-MAX2SAT` gap problem is NP-hard to approximate for problems with $I = 2016n$ clauses, $\gamma^- = (2011+\varepsilon)n$ lower boundary, and $\gamma^+ = (2012-\varepsilon)n$ upper boundary. We can use Lemmas 1 and 2 to prove a reduction from this gap problem to `D-2-TOK`'s gap problem. Notably, our reduction equates $\delta = 329J + 3I - \gamma$, for both γ^- and γ^+. Analysing the gap of the resulting tokenisation problem, we find that this problem is thus NP-hard for an approximation ratio $\frac{\delta^-}{\delta^+}$ of at least 1.000002. This implies that no polynomial-time algorithm can approximate the direct binary tokenisation optimisation problem with an approximation ratio better than this constant, unless P = NP. □

While the constant above (i.e., 1.000002) is remarkably small, we note that our proof makes no attempt to optimise this bound. Our lemma's main takeaway is that it is not possible to compute `D-2-TOK` with approximation ratios arbitrarily close to 1 in polynomial time. Other larger bounds likely exist and, in fact, it might even be possible that there is no constant-factor approximation for `D-2-TOK` at all.

4.3 Bottom-Up Binary Tokenisation is Hard to Decide and Approximate

This section addresses the computational hardness of finding an optimal merge sequence in bottom-up binary tokenisation. We establish that the problem is NP-hard (in §4.3.1). Furthermore, we prove that the problem is also hard to approximate; specifically, we show there is no `ptas` for it, unless P = NP (in §4.3.2). As for the direct case, our argument proceeds by first proving the hardness of the *decision problem*, and then leveraging this result to demonstrate the hardness of a corresponding gap problem.

4.3.1 The Bottom-Up Binary Tokenisation Problem is NP-Complete

As before, we use a reduction from `3-OCC-MAX2SAT` to prove this problem's NP-hardness.

Reduction 2. *Consider an instance of the* `3-OCC-MAX2SAT` *decision problem and a binary alphabet* $\Sigma = \{0,1\}$*. Again, for each variable* X_j*, let* $\mathtt{x}_j^\mathtt{T} = 0^{2j-1}$ *and* $\mathtt{x}_j^\mathtt{F} = 0^{2j}$*. Then we build subdatasets:*

$$\mathcal{D}_1 = \{11, \mathtt{x}_j^\mathtt{T}, \mathtt{x}_j^\mathtt{F}, 1\mathtt{x}_j^\mathtt{T}, \mathtt{x}_j^\mathtt{T}1, 1\mathtt{x}_j^\mathtt{F}, \mathtt{x}_j^\mathtt{F}1, \mathtt{x}_j^\mathtt{T}11, 11\mathtt{x}_j^\mathtt{F}\} \times f \qquad \mathcal{D}_2 = \{1\mathtt{x}_j^\mathtt{T}1, 1\mathtt{x}_j^\mathtt{F}1, 1\mathtt{x}_j^\mathtt{T}11, 11\mathtt{x}_j^\mathtt{F}1\} \times f' \tag{8a}$$

$$\mathcal{D}_3 = \{1\mathtt{x}_j^\mathtt{T}1\mathtt{x}_j^\mathtt{F}1, 11\mathtt{x}_j^\mathtt{F}1\mathtt{x}_j^\mathtt{T}11\} \times f'' \qquad \mathcal{D}_4 = \{1\mathtt{x}_j^\mathtt{F}1\mathtt{x}_j^\mathtt{T}11, 11\mathtt{x}_j^\mathtt{F}1\mathtt{x}_j^\mathtt{T}1\} \times f''' \tag{8b}$$

$$\mathcal{D}_5 = \begin{Bmatrix} 1\mathtt{x}_j^\mathtt{T}1\mathtt{x}_{j'}^\mathtt{F}1 & \texttt{if } L_i^1 = X_j \texttt{ and } L_i^2 = \neg X_{j'} \\ 1\mathtt{x}_{j'}^\mathtt{T}1\mathtt{x}_j^\mathtt{F}1 & \texttt{if } L_i^1 = \neg X_j \texttt{ and } L_i^2 = X_{j'} \\ 11\mathtt{x}_j^\mathtt{F}1\mathtt{x}_{j'}^\mathtt{F}1 & \texttt{if } L_i^1 = \neg X_j \texttt{ and } L_i^2 = \neg X_{j'} \\ 1\mathtt{x}_j^\mathtt{T}1\mathtt{x}_{j'}^\mathtt{T}11 & \texttt{if } L_i^1 = X_j \texttt{ and } L_i^2 = X_{j'} \end{Bmatrix} \times 1 \tag{8c}$$

These subdataset multiplicities are $f''' \stackrel{\text{def}}{=} 4$, $f'' \stackrel{\text{def}}{=} 2(2f'''+3)+1 = 23$, $f' \stackrel{\text{def}}{=} 2(2f''+2f'''+3)+1 = 115$, $f \stackrel{\text{def}}{=} 2(2f'+2f''+2f'''+3)+1 = 575$. *We set the vocabulary size to* $K = 10J$ *and the target compressed length to* $\delta = (8J+1)f+6Jf'+4Jf''+4Jf'''+3I-\gamma = 5398J+575+3I-\gamma$.

We write $\mathrm{R2}(\mathcal{X},\mathcal{C},\gamma)$ to represent $(\mathcal{D},K,\delta)$, the B-2-TOK instance constructed by this reduction. As before, this is a polynomial-timed reduction. We now prove the equivalence $\mathrm{3OM2S}(\mathcal{X},\mathcal{C},\gamma) \iff \mathrm{Tok}^2_{\uparrow}(\mathrm{R2}(\mathcal{X},\mathcal{C},\gamma))$ which shows the reduction's correctness and that that B-2-TOK is NP-hard.

Theorem 3. *The bottom-up binary tokenisation decision problem is* NP*-complete.*

Proof sketch. This proof is done in two steps.

Forward step ($\mathrm{3OM2S}(\mathcal{X},\mathcal{C},\gamma) \implies \mathrm{Tok}^2_{\uparrow}(\mathrm{R1}(\mathcal{X},\mathcal{C},\gamma))$)**.** See full proof in Lemma 5 in App. D. Assume the 3-OCC-MAX2SAT instance admits an assignment $\mathcal{X}^\star = \{x^\star_j\}_{j=1}^J$ satisfying at least γ clauses. We construct a merge sequence $\mathbf{m} = \mathbf{m}_1 \circ \mathbf{m}_2 \circ \mathbf{m}_3 \circ \mathbf{m}_4 \circ \mathbf{m}_5 \circ \mathbf{m}_6$, where $\mathbf{m}_1, \mathbf{m}_2, \mathbf{m}_4, \mathbf{m}_6$ are **structural merges** that appear in every valid tokeniser solution and ensure that all variable and literal gadgets are properly compressed; $\mathbf{m}_3, \mathbf{m}_5$ are **assignment-dependent merges**, chosen according to x: for each variable $x^\star_j$, we merge $\langle 1, \mathbf{x}^{\mathtt{T}}_j 11\rangle, \langle 1\mathbf{x}^{\mathtt{T}}_j, 1\rangle$ if $x^\star_j = \mathtt{T}$, and $\langle 11\mathbf{x}^{\mathtt{F}}_j, 1\rangle, \langle 1, \mathbf{x}^{\mathtt{F}}_j 1\rangle$ otherwise. Applying $\mathbf{m}$ to the string sets $\mathcal{D}_1, \mathcal{D}_2, \mathcal{D}_3, \mathcal{D}_4$ gives the fixed compressed length $5398J + 575$. For the strings $\mathcal{D}_5$, the construction ensures that each clause compresses to 2 tokens if at least one of its two literals is true under $\mathcal{X}^\star$, and remains at 3 tokens otherwise. Since $\mathcal{X}^\star$ satisfies at least γ clauses, we obtain at most $3I - \gamma$ symbols. Compression thus satisfies the budget δ.

Backward step ($\mathrm{Tok}^2_{\uparrow}(\mathrm{R2}(\mathcal{X},\mathcal{C},\gamma)) \implies \mathrm{3OM2S}(\mathcal{X},\mathcal{C},\gamma)$)**.** See full proof in Lemma 6 in App. E. We consider sat-compliant *direct* tokenisers, which must contain all tokens of the form $11, 11, \mathbf{x}^{\mathtt{T}}_j, \mathbf{x}^{\mathtt{F}}_j, 1\mathbf{x}^{\mathtt{T}}_j, \mathbf{x}^{\mathtt{T}}_j 1, 1\mathbf{x}^{\mathtt{F}}_j, \mathbf{x}^{\mathtt{F}}_j 1, \mathbf{x}^{\mathtt{T}}_j 11, 11\mathbf{x}^{\mathtt{F}}_j$; and contain either $1\mathbf{x}^{\mathtt{T}}_j 1, 1\mathbf{x}^{\mathtt{T}}_j 11$ or $1\mathbf{x}^{\mathtt{F}}_j 1, 11\mathbf{x}^{\mathtt{F}}_j 1$ for each $j \in \{1,\ldots,J\}$. These conditions are enforced by datasets $\mathcal{D}_1$ to $\mathcal{D}_4$. As before, if such a tokeniser achieves the desired compression, it must correspond to an assignment $\mathcal{X}^\star$ which satisfies the desired number of clauses. To finish, we show that a sat-compliant direct tokeniser always corresponds to a bottom-up tokeniser with the same compression quality. □

4.3.2 The Bottom-Up Binary Tokenisation Gap Problem is NP-hard

As in §4.2.2, we perform a reduction from a gap variant to show hardness of approximation.

Theorem 4. *The bottom-up binary tokenisation gap problem is* NP*-hard. Thus, the bottom-up binary tokenisation optimisation problem is not in* PTAS*, unless* P = NP.

Proof sketch. See proof in Lemma 7 in App. F. A similar proof to Theorem 2 applies here, except with different values. We find that no polynomial-time algorithm can solve the bottom-up binary tokenisation optimisation problem with an approximation ratio better than 1.0000001, unless P = NP. □

5 Unary Tokenisation is Hard to Decide

We now move on to the unary tokenisation case. Here, we work with alphabets composed of a single symbol, i.e.: $\Sigma = \{a\}$. As $\Sigma^* = \{a^\ell \mid \ell \in \mathbb{N}\}$, it follows that unary character-strings $\mathbf{c} \in \Sigma^*$ may only differ from one another in their length. There exists thus an isomorphism (given by the function $|\cdot|$ and its inverse) between these character-strings and their **string-lengths**, $\ell \in \mathbb{N}$. A natural notation for such problems is then to work directly with string-lengths. In this section, we will thus represent a character-string $\mathbf{c} \in \Sigma^*$ by its length $\ell \in \mathbb{N}$; a dataset $\mathcal{D} = \{\mathbf{c}_m\}_{m=1}^M$ by the lengths of its strings $\mathcal{D}_{\mathbb{N}} = \{\ell_m\}_{m=1}^M$, where $\mathbf{c}_m = a^{\ell_m}$; and a vocabulary $\mathcal{S} \subset \Sigma^+$ by a set of string-lengths $\mathcal{S}_{\mathbb{N}} \subset \mathbb{N}_+$. A subword-string is then a sequence of such string-lengths, $\mathbf{s}_{\mathbb{N}} \in \mathcal{L}^*$, and we have:

$$\mathtt{detok}(\mathbf{s}_{\mathbb{N}}) \stackrel{\text{def}}{=} \mathtt{sum}(\mathbf{s}_{\mathbb{N}}) \qquad \mathtt{tok}_\diamond[\mathcal{S}_{\mathbb{N}}](\ell) \stackrel{\text{def}}{=} \operatorname*{arg\,min}_{\mathbf{s}_{\mathbb{N}} \in \mathcal{S}_{\mathbb{N}}{}^*} |\mathbf{s}_{\mathbb{N}}|, \text{ s.t., } \ell = \mathtt{sum}(\mathbf{s}_{\mathbb{N}}) \tag{9}$$

where we overload the functions detok and $\mathtt{tok}_\diamond$ to handle this unary-strings representation. Note that all these definitions are equivalent (up to an isomorphism) to the definitions in §3.

When posing either the optimisation or decision version of the unary tokenisation problems, we could thus work with either representation of our data (as strings or string-lengths) and the solutions must

be the same. However, the complexity of an algorithm is typically measured as a function of the length of its input. If this input is a unary string, the input will be as long as this string's length. If this input is a number, however, this input's length behaves logarithmically on the value of the number itself (as this number would typically be encoded in a compact binary representation). When dealing with problems such as unary tokenisation, this introduces an important subtlety: the problem's complexity status may change depending on how we represent it (with strings or string-lengths). If such a problem is NP-hard when either representation is given, it is called **strongly NP-hard**. If this problem is NP-hard only in its string-length representation, but not when represented using unary strings, it is **weakly NP-hard**.[7] Importantly, Fact 1 applies only to strongly NP-hard unary problems; as the trivial identity we use in its proof would not be valid for unary problems with string-length representations. For unary tokenisation, the unary representation (where strings are explicitly represented) is more natural, and we are thus interested in strong NP-hardness.

5.1 Direct Unary Tokenisation is Strongly NP-Complete

In this section, we prove that the direct unary tokenisation problem is strongly NP-complete. In App. G, we prove that the problem is in NP. To prove NP-hardness of direct unary tokenization, we then design a polynomial-time reduction from the well-known vertex cover problem (`vertex-cover`).[8] Let $(\mathcal{V}, \mathcal{E})$ represent a finite, simple, undirected graph with $\mathcal{V} = \{v_1, \ldots, v_J\}$ and $\mathcal{E} \subseteq \{(v, v') \mid v, v' \in \mathcal{V}, v \neq v'\}$. A set $\mathcal{C} \subseteq \mathcal{V}$ is a **vertex cover** if for every edge $(v, v') \in \mathcal{E}$ we have that either v or v' is in $\mathcal{C}$. Given a budget $\psi \in \mathbb{N}$, the vertex cover problem requires deciding whether a graph has a vertex cover with at most ψ vertices.

Definition 3. *Given a graph $(\mathcal{V}, \mathcal{E})$ and a budget $\psi \in \mathbb{N}$, the **vertex cover decision problem** asks whether there exists a vertex cover $\mathcal{C} \subseteq \mathcal{V}$ with $|\mathcal{C}| \leq \psi$ in this graph.*

For convenience, we will write $\mathrm{VC}(\mathcal{V}, \mathcal{E}, \psi)$ for a function which returns `T` if its input is a satisfiable instance of the `vertex-cover` decision problem, and `F` otherwise. We now provide a polynomial-time reduction from `vertex-cover` to `D-1-TOK`, which will prove `D-1-TOK`'s NP-hardness.

Reduction 3. *Consider an instance $(\mathcal{V}, \mathcal{E}, \psi)$ of `vertex-cover` and let $N \stackrel{\text{def}}{=} (J + I + 1)^3$, where $J = |\mathcal{V}|$ and $I = |\mathcal{E}|$. Now, let $\mathtt{enc}(v_j) = j + j^2 N + j^3 N^2$ and $B = N^4$. We construct three subdatasets from this graph as:*

$$\mathcal{D}_1 = \{\ell_j \mid v_j \in \mathcal{V}\} \cup \{B\}, \quad \text{where } \ell_j = \mathtt{enc}(v_j) \qquad \textit{vertex strings} \qquad (10a)$$

$$\mathcal{D}_2 = \{\ell'_j \mid v_j \in \mathcal{V}\}, \quad \text{where } \ell'_j = \mathtt{enc}(v_j) + B \qquad \textit{cover strings} \qquad (10b)$$

$$\mathcal{D}_3 = \{\ell''_{j,j'} \mid (v_j, v_{j'} \in \mathcal{E})\}, \quad \text{where } \ell''_{j,j'} = \mathtt{enc}(v_j) + \mathtt{enc}(v_{j'}) + B \qquad \textit{edge strings} \qquad (10c)$$

Finally, we merge these subdatasets to form a dataset $\mathcal{D} = \mathcal{D}_1 \cup \mathcal{D}_2 \cup \mathcal{D}_3$, and set $K = J + 1 + \psi$ and $\delta = 3J + 2I + 1 - \psi$.

As before, we complete our NP-hardness proof by showing this to be a valid reduction, i.e., that: $\mathrm{VC}(\mathcal{V}, \mathcal{E}, \psi) \iff \mathrm{Tok}^1_{\$}(\mathrm{R3}(\mathcal{V}, \mathcal{E}, \psi))$. Notably, our reduction outputs (in polynomial time) an instance of the direct unary tokenisation problem in string-length form. As such, by proving the correctness of this reduction, we prove the *strong* NP-hardness of `D-1-TOK`.

Theorem 5. *The direct unary tokenisation decision problem is strongly NP-complete.*

Proof sketch. This proof is done in two steps.

Forward step ($\mathrm{VC}(\mathcal{V}, \mathcal{E}, \psi) \implies \mathrm{Tok}^1_{\$}(\mathrm{R3}(\mathcal{V}, \mathcal{E}, \psi))$). See full proof in Lemma 9 in App. H. Suppose that the given instance of `vertex-cover` is true, i.e., that $\mathrm{VC}(\mathcal{V}, \mathcal{E}, \psi) = \mathtt{T}$. Now, let $\mathcal{C}^\star \subseteq \mathcal{V}$ be a vertex cover which satisfies this instance. Then we can build a tokeniser with vocabulary: $\mathcal{S}_{\mathbb{N}} = \{\ell_j \mid v_j \in \mathcal{V}\} \cup \{B\} \cup \{\ell'_j \mid v_j \in \mathcal{C}^\star\}$. This tokeniser will encode: all strings in $\mathcal{D}_1$ as a single symbol; ψ strings in $\mathcal{D}_2$ with a single symbol and others with 2; and all strings in $\mathcal{D}_3$ with two symbols (as, per our assumption, all edges have at least one vertex in $\mathcal{C}^\star$). This means the total amount of tokens used is: $(J + 1) + (2J - \psi) + 2I = \delta$. Therefore, $\mathrm{Tok}^1_{\$}(\mathcal{D}, K, \delta) = \mathtt{T}$.

[7] Note that the opposite case—where a problem is NP-hard only when representing the data as strings, but not strings-lengths—is not possible, as strings have a larger size than their lengths.

[8] The NP-hardness of `vertex-cover` was proven by Karp (1972) in his groundbreaking paper introducing the very concept of NP-hardness, and can be found in the textbook by Garey and Johnson (1979).

Backward step ($\text{Tok}^1_{\Rightarrow}(\text{R3}(\mathcal{V}, \mathcal{E}, \psi)) \implies \text{VC}(\mathcal{V}, \mathcal{E}, \psi)$)**.** See full proof in Lemma 10 in App. I. We prove this lemma in 4 steps. First, we show that all string-lengths in $\mathcal{D}_{\mathbb{N}}$ are unique. Second, we show that an optimal tokeniser's vocabulary must contain only full strings in $\mathcal{D}_{\mathbb{N}}$. Third, we show that an optimal tokeniser's vocabulary must include all strings in $\mathcal{D}_1$. Fourth, we show that if a compression of δ is achieved, than this `vertex-cover` instance must be true. Notably, three of these steps rely on the fact that we can use N as a numerical base to prove the uniqueness of both: (i) individual string-lengths, as well as (ii) their pair-wise summed values. □

Interestingly, the direct unary tokenisation problem is tightly related to the field of choosing denominations for a coin system. In fact, the application of the function $\texttt{tok}_{\Rightarrow}[\mathcal{S}_{\mathbb{N}}](\ell)$ is equivalent to the change-making problem; a problem shown to be (weakly) NP-hard by Lueker (1975). (Note that this problem is only weakly NP-hard, as we can solve it in polynomial time when the input is given in strings-form.) The direct unary tokenisation problem can thus be equivalently seen as a **general optimal denomination problem**, where—given a set of common currency transactions—one must select optimal coin denominations for a currency; see Shallit (2003) for a discussion of this problem.

Corollary 1. *The general optimal denomination decision problem is strongly* NP*-complete.*

5.2 A Variant of Bottom-up Unary Tokenisation is (at Least) Weakly NP-Hard

While direct tokenisation over a unary alphabet is strongly NP-complete, our current picture of the complexity of its bottom-up counterpart is more nuanced. In bottom-up tokenisation, one must find a merge sequence $\mathbf{m}$ which is then applied (by $\texttt{tok}_{\uparrow}[\mathbf{m}](\mathbf{c})$) *exhaustively* and *in sequence*, replacing all occurrences of each pair one at a time. A variant of this problem—termed **optimal pair encoding (OPE) tokenisation**—relaxes this requirement, using the merge sequence for a different purpose: to define a **merge-extracted vocabulary** $\mathcal{S}_{\mathbf{m}} = \Sigma \cup \{s_1 \circ s_2 \mid m \in \mathbf{m}, m = (s_1, s_2)\}$. The final tokenization is then produced by optimally applying this vocabulary, which can be done using the direct encoding function ($\texttt{tok}_{\Rightarrow}[\mathcal{S}_{\mathbf{m}}](\mathbf{c})$). This approach thus ensures that a merge is used only if it contributes to the most efficient segmentation overall. Notably, this variation was used by Schmidt et al. (2024) and formally analysed by Kozma and Voderholzer (2024). Now, let the **OPE unary tokenisation problem** be defined similarly to the other n-ary tokenisation problems (in Definition 1), but while constraining the search space to the set of OPE tokenisers: $\mathcal{T}_{\text{ope}} \stackrel{\text{def}}{=} \{\texttt{tok}_{\Rightarrow}[\mathcal{S}_{\mathbf{m}}] \mid \mathbf{m} \in \mathcal{M}^*\}$. Having defined the decision problem, we now establish its computational hardness.

Theorem 6. *The unary optimal-pair-encoding decision problem is weakly* NP*-complete.*

Proof sketch. The full proof can be found in App. K. Inclusion in NP follows from Kozma and Voderholzer (2024). The proof of NP-hardness is achieved via a polynomial-time reduction from the addition chain sequence decision problem (see App. J for a formal definition), which is known to be NP-complete when its input numbers are encoded in binary (Downey et al., 1981). The reduction reveals a natural connection between the two problems: finding the shortest addition chain for a set of numbers is equivalent to a special case of unary OPE where every string in the dataset must be compressed into a single token. □

6 Conclusion and Limitations

We provided several hardness results on bottom-up and direct tokenisation with bounded alphabets, thus answering open questions posed by both Kozma and Voderholzer (2024) and Whittington et al. (2025). A number of open questions remain, however, in particular with respect to approximability. For instance, while we showed that the direct binary optimisation problem cannot be approximated arbitrarily well (unless P = NP)—and while it seems likely that the lower bound provided in the proof of Lemma 3 can be significantly lifted—it is unclear whether any constant approximation ratio can even be obtained. With respect to decision problems, while we showed strong NP-hardness of direct unary tokenisation, we were so far only able prove: (i) weak NP-hardness of OPE unary tokenisation, and (ii) no hardness result for (standard) bottom-up unary tokenisation. Finally, the results of our work are limited in that we consider (i) compression as objective, and (ii) bottom-up and direct tokenization only; the hardness of both other objectives and variants remains open. Overall, however, our results show that tokenisation remains a hard problem, even when restricted to small (even binary) alphabets. Future work should thus explore provably good approximation algorithms.

REFERENCES

Mehdi Ali, Michael Fromm, Klaudia Thellmann, Richard Rutmann, Max Lübbering, Johannes Leveling, Katrin Klug, Jan Ebert, Niclas Doll, Jasper Buschhoff, Charvi Jain, Alexander Weber, Lena Jurkschat, Hammam Abdelwahab, Chelsea John, Pedro Ortiz Suarez, Malte Ostendorff, Samuel Weinbach, Rafet Sifa, Stefan Kesselheim, and Nicolas Flores-Herr. 2024. Tokenizer choice for LLM training: Negligible or crucial? In *Findings of the Association for Computational Linguistics: NAACL 2024*, pages 3907–3924, Mexico City, Mexico. Association for Computational Linguistics.

Piotr Berman and Marek Karpinski. 1998. On some tighter inapproximability results, further improvements. In *Electronic Colloquium on Computational Complexity*, volume 65.

Piotr Berman and Marek Karpinski. 1999. On some tighter inapproximability results (extended abstract). In *Automata, Languages and Programming — 26th International Colloquium, ICALP 1999, Proceedings*, volume 1644 of *Lecture Notes in Computer Science*, pages 200–209, Berlin, Heidelberg. Springer.

Peter Downey, Benton Leong, and Ravi Sethi. 1981. Computing sequences with addition chains. *SIAM Journal on Computing*, 10(3):638–646.

Matthias Gallé. 2019. Investigating the effectiveness of BPE: The power of shorter sequences. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 1375–1381, Hong Kong, China. Association for Computational Linguistics.

Michael R. Garey and David S. Johnson. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, San Francisco.

Omer Goldman, Avi Caciularu, Matan Eyal, Kris Cao, Idan Szpektor, and Reut Tsarfaty. 2024. Unpacking tokenization: Evaluating text compression and its correlation with model performance. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 2274–2286, Bangkok, Thailand. Association for Computational Linguistics.

Thamme Gowda and Jonathan May. 2020. Finding the optimal vocabulary size for neural machine translation. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 3955–3964, Online. Association for Computational Linguistics.

Richard M. Karp. 1972. Reducibility among combinatorial problems. In Raymond E. Miller and James W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, New York.

László Kozma and Johannes Voderholzer. 2024. Theoretical analysis of byte-pair encoding. *Preprint*, arXiv:2411.08671.

Taku Kudo. 2018. Subword regularization: Improving neural network translation models with multiple subword candidates. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 66–75, Melbourne, Australia. Association for Computational Linguistics.

Jia Peng Lim, Davin Choo, and Hady W. Lauw. 2025. A partition cover approach to tokenization. *Preprint*, arXiv:2501.06246.

G.S. Lueker. 1975. *Two NP-complete Problems in Nonnegative Integep Programming*. Technical report: Computer Science Laboratory. Univ.

Phillip Rust, Jonas Pfeiffer, Ivan Vulić, Sebastian Ruder, and Iryna Gurevych. 2021. How good is your tokenizer? On the monolingual performance of multilingual language models. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 3118–3135, Online. Association for Computational Linguistics.

Craig W. Schmidt, Varshini Reddy, Haoran Zhang, Alec Alameddine, Omri Uzan, Yuval Pinter, and Chris Tanner. 2024. Tokenization is more than compression. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 678–702, Miami, Florida, USA. Association for Computational Linguistics.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1715–1725, Berlin, Germany. Association for Computational Linguistics.

Jeffrey Shallit. 2003. What this country needs is an 18c piece. In *Math. Intelligencer 25(2)*.

Omri Uzan, Craig W. Schmidt, Chris Tanner, and Yuval Pinter. 2024. Greed is all you need: An evaluation of tokenizer inference methods. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 813–822, Bangkok, Thailand. Association for Computational Linguistics.

Philip Whittington, Gregor Bachmann, and Tiago Pimentel. 2025. Tokenisation is NP-complete. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Vienna, Austria. Association for Computational Linguistics.

Vilém Zouhar, Clara Meister, Juan Gastaldi, Li Du, Mrinmaya Sachan, and Ryan Cotterell. 2023a. Tokenization and the noiseless channel. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 5184–5207, Toronto, Canada. Association for Computational Linguistics.

Vilém Zouhar, Clara Meister, Juan Gastaldi, Li Du, Tim Vieira, Mrinmaya Sachan, and Ryan Cotterell. 2023b. A formal perspective on byte-pair encoding. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 598–614, Toronto, Canada. Association for Computational Linguistics.

A Proof of Lemma 1

Lemma 1. *If a* 3-OCC-MAX2SAT *instance is satisfiable, then the* D-2-TOK *instance output by Reduction 1 is also satisfiable. Formally:* $\mathrm{3OM2S}(\mathcal{X},\mathcal{C},\gamma) \implies \mathrm{Tok}^2_{\diamond}(\mathrm{R1}(\mathcal{X},\mathcal{C},\gamma))$.

Proof. Assume this $(\mathcal{X},\mathcal{C},\gamma)$ instance of the 3-OCC-MAX2SAT problem is satisfiable, i.e., that $\mathrm{3OM2S}(\mathcal{X},\mathcal{C},\gamma)$ is true. We must prove that, in this case, $\mathrm{Tok}^2_{\diamond}(\mathrm{R1}(\mathcal{X},\mathcal{C},\gamma))$ is also true. Now, let $\mathcal{X}^\star = \{x_j^\star\}_{j=1}^J$ be any satisfying solution to the $(\mathcal{X},\mathcal{C},\gamma)$ instance. We will denote the number of clauses satisfied by $\mathcal{X}^\star$ by $\gamma^\star$, noting that $\gamma^\star \geq \gamma$ by assumption. We can construct a tokeniser from this solution as follows:

$$\mathcal{S} = \Sigma \bigcup \left\{ 1\mathtt{x}_j^{\mathtt{T}},\ \mathtt{x}_j^{\mathtt{T}}1,\ 1\mathtt{x}_j^{\mathtt{F}},\ \mathtt{x}_j^{\mathtt{F}}1 \right\}_{j=1}^{J} \quad \bigcup \quad \left\{ 1\mathtt{x}_j^{\mathtt{T}}1 \ \texttt{if}\ x_j^\star = \mathtt{T}\ \texttt{else}\ 1\mathtt{x}_j^{\mathtt{F}}1 \right\}_{j=1}^{J} \tag{11}$$

Note that—as required by our reduction—this tokeniser has vocabulary size $|\mathcal{S}| = |\Sigma| + K$, since $K = 5J$ tokens were added. Under this tokeniser, we have:

$$\mathtt{tok}_{\diamond}[\mathcal{S}](\mathcal{D}_1) = \left\{ \langle 1\mathtt{x}_j^{\mathtt{T}}\rangle,\ \langle \mathtt{x}_j^{\mathtt{T}}1\rangle,\ \langle 1\mathtt{x}_j^{\mathtt{F}}\rangle,\ \langle \mathtt{x}_j^{\mathtt{F}}1\rangle \quad \text{(length 1)} \qquad \middle|\ 1 \leq j \leq J \right\} \quad \times f \tag{12a}$$

$$\mathtt{tok}_{\diamond}[\mathcal{S}](\mathcal{D}_2) = \left\{ \begin{array}{lll} \langle 1\mathtt{x}_j^{\mathtt{T}}1\rangle,\ \langle 1\mathtt{x}_j^{\mathtt{F}},\ 1\rangle & \text{(length 3)} & \texttt{if}\ x_j^\star = \mathtt{T} \\ \langle 1\mathtt{x}_j^{\mathtt{T}},\ 1\rangle,\ \langle 1\mathtt{x}_j^{\mathtt{F}}1\rangle & \text{(length 3)} & \texttt{else} \end{array} \middle|\ 1 \leq j \leq J \right\} \quad \times f' \tag{12b}$$

$$\mathtt{tok}_{\diamond}[\mathcal{S}](\mathcal{D}_3) = \left\{ \begin{array}{lll} \langle 1\mathtt{x}_j^{\mathtt{T}}1,\ \mathtt{x}_j^{\mathtt{F}}1\rangle & \text{(length 2)} & \texttt{if}\ x_j^\star = \mathtt{T} \\ \langle 1\mathtt{x}_j^{\mathtt{T}},\ 1\mathtt{x}_j^{\mathtt{F}}1\rangle & \text{(length 2)} & \texttt{else} \end{array} \middle|\ 1 \leq j \leq J \right\} \quad \times f'' \tag{12c}$$

$$\mathtt{tok}_{\diamond}[\mathcal{S}](\mathcal{D}_4) = \left\{ \begin{array}{lll} \langle 1L_i^1 1,\ L_i^2 1\rangle & \text{(length 2)} & \texttt{if}\ L_i^1 = \mathtt{T} \\ \langle 1L_i^1,\ 1L_i^2 1\rangle & \text{(length 2)} & \texttt{elif}\ L_i^2 = \mathtt{T} \\ \langle 1L_i^1,\ 1,\ L_i^2 1\rangle & \text{(length 3)} & \texttt{else} \end{array} \middle|\ 1 \leq i \leq I \right\} \quad \times 1 \tag{12d}$$

where we override function $\mathtt{tok}_{\diamond}[\mathcal{S}]$ to apply elementwise to a full dataset of character-strings, instead of to a unique $\mathbf{c}$. Consequently, we get the compressed lengths:

$$\mathfrak{G}_\ell(\mathtt{tok}_{\diamond}[\mathcal{S}],\mathcal{D}_1) = 4\,J\,f = 252\,J\,f, \qquad \mathfrak{G}_\ell(\mathtt{tok}_{\diamond}[\mathcal{S}],\mathcal{D}_2) = 3\,J\,f' = 63\,J, \tag{13a}$$

$$\mathfrak{G}_\ell(\mathtt{tok}_{\diamond}[\mathcal{S}],\mathcal{D}_3) = 2\,J\,f'' = 14\,J, \qquad \mathfrak{G}_\ell(\mathtt{tok}_{\diamond}[\mathcal{S}],\mathcal{D}_4) = 3\,I - \gamma^\star \tag{13b}$$

We have that each character-string in dataset $\mathcal{D}_4$ is compressed to 2 symbols if either L_i^1 or L_i^2 are true, and else is kept at 3 symbols; the $\gamma^\star$ satisfied clauses in $\mathcal{X}^\star$ will thus be compressed to 2 symbols and the unsatisfied clauses to 3. Summing these values together, we get the compressed length of the entire dataset under this tokeniser: $\mathfrak{G}_\ell(\mathtt{tok}_{\diamond}[\mathcal{S}],\mathcal{D}) = 329I + 3\,I - \gamma^\star$. Finally:

$$\gamma^\star \geq \gamma \implies 329I + 3\,I - \gamma^\star \leq 329I + 3\,I - \gamma \tag{14}$$

This completes this proof. □

B Proof of Lemma 2

Before starting our lemma's proof, we give a few definitions which will be useful throughout it. First, we define a **sat-compliant tokeniser** to be any tokeniser which: (i) contains all tokens of the form $1\mathtt{x}_j^{\mathtt{T}}, \mathtt{x}_j^{\mathtt{T}}1, 1\mathtt{x}_j^{\mathtt{F}}, \mathtt{x}_j^{\mathtt{F}}1$; and (ii) contains either $1\mathtt{x}_j^{\mathtt{T}}1$ or $1\mathtt{x}_j^{\mathtt{F}}1$ for each $j \in \{0,\ldots,J\}$. Otherwise, we call the tokeniser **sat-noncompliant**. Given the vocabulary of a sat-compliant tokeniser, we can easily build an assignment to a 3-OCC-MAX2SAT instance with the following function:

$$g(\mathcal{S}) = \{x_j^\star\}_{j=1}^J, \ \texttt{where} \left\{ \begin{array}{ll} x_j^\star = \mathtt{T} & \texttt{if}\ 1\mathtt{x}_j^{\mathtt{T}}1 \in \mathcal{S} \\ x_j^\star = \mathtt{F} & \texttt{elif}\ 1\mathtt{x}_j^{\mathtt{F}}1 \in \mathcal{S} \end{array} \right. \tag{15}$$

Further, we will define as a **101-string** any character-string of the form 10^+1, and as a **10101-string** any character-string of the form 10^+10^+1. (The 0^+ notation stands for a sequence of one or more 0 characters.) Considering the datasets output by Reduction 1, we know that there are no 101-strings in dataset $\mathcal{D}_1$. Further, we know that each unique 101-string appears in datasets $\mathcal{D}_2$ and $\mathcal{D}_3$ exactly f' and f'' times, respectively, and exactly 3 times in $\mathcal{D}_4$. (This is due to us working with the three-occurrences variant of MAX2SAT and to the fact that $\mathbf{x}_j^{\mathtt{T}} = 0^{2j-1}$ and $\mathbf{x}_j^{\mathtt{F}} = 0^{2j}$.) We now prove our lemma.

Lemma 2. *If the* D-2-TOK *instance output by Reduction 1 is satisfiable, then the* 3-OCC-MAX2SAT *instance which generated it is as well. Formally:* $\mathrm{Tok}^2_{\diamondsuit}(\mathrm{R1}(\mathcal{X},\mathcal{C},\gamma)) \implies \mathrm{3OM2S}(\mathcal{X},\mathcal{C},\gamma)$.

Proof. Assume this $(\mathcal{D}, K, \delta)$ instance of D-2-TOK—where $(\mathcal{D}, K, \delta) = \mathrm{R1}(\mathcal{X},\mathcal{C},\gamma)$—is satisfiable, i.e., that $\mathrm{Tok}^2_{\diamondsuit}(\mathrm{R1}(\mathcal{X},\mathcal{C},\gamma))$ evaluates to true. We must prove that, in this case, $\mathrm{3OM2S}(\mathcal{X},\mathcal{C},\gamma)$ also evaluates to true. Now, let $\mathcal{S}_{\texttt{opt}}$ be the optimal solution to the $(\mathcal{D}, K, \delta)$ instance. We know, by definition, that:

$$\mathrm{Tok}^2_{\diamondsuit}(\mathrm{R1}(\mathcal{X},\mathcal{C},\gamma)) \iff \Big(\mathfrak{G}_\ell(\texttt{tok}_{\diamondsuit}[\mathcal{S}_{\texttt{opt}}],\mathcal{D}) \leq \delta\Big) \tag{16}$$

We can thus prove this lemma by showing the following implication:

$$\Big(\mathfrak{G}_\ell(\texttt{tok}_{\diamondsuit}[\mathcal{S}_{\texttt{opt}}],\mathcal{D}) \leq \delta\Big) \implies \mathrm{3OM2S}(\mathcal{X},\mathcal{C},\gamma) \tag{17}$$

We will now prove this lemma in four steps:

① we prove that $\mathcal{S}_{\texttt{opt}}$ must include all tokens of the form $1\mathbf{x}^{\mathrm{T}}_j, \mathbf{x}^{\mathrm{T}}_j 1, 1\mathbf{x}^{\mathrm{F}}_j, \mathbf{x}^{\mathrm{F}}_j 1$;

② we prove that $\mathcal{S}_{\texttt{opt}}$ must, in addition to the tokens above, only include tokens of the form $1\mathbf{x}^{\mathrm{T}}_j 1, 1\mathbf{x}^{\mathrm{F}}_j 1$;

③ we prove that $\mathcal{S}_{\texttt{opt}}$ may only include, for each j, either token $1\mathbf{x}^{\mathrm{T}}_j 1$ or $1\mathbf{x}^{\mathrm{F}}_j 1$;

④ finally, we prove that, if $\Big(\mathfrak{G}_\ell(\texttt{tok}_{\diamondsuit}[\mathcal{S}_{\texttt{opt}}],\mathcal{D}) \leq \delta\Big)$, we can build a variable assignment which satisfies this $(\mathcal{X},\mathcal{C},\gamma)$ 3-OCC-MAX2SAT instance.

Note that, together, steps one to three show that $\mathcal{S}_{\texttt{opt}}$ must be the vocabulary of a sat-compliant tokeniser; in step four, we will then rely on the function g (defined above) to convert this vocabulary into a satisfying assignment $\varkappa = g(\mathcal{S}_{\texttt{opt}})$ to 3-OCC-MAX2SAT. □

LemmaProofStep 1. (Step ①). *An optimal tokeniser must include all tokens of the form* $1\mathbf{x}^{\mathrm{T}}_j, \mathbf{x}^{\mathrm{T}}_j 1, 1\mathbf{x}^{\mathrm{F}}_j, \mathbf{x}^{\mathrm{F}}_j 1$*, i.e.,:*

$$\Big\{1\mathbf{x}^{\mathrm{T}}_j, \mathbf{x}^{\mathrm{T}}_j 1, 1\mathbf{x}^{\mathrm{F}}_j, \mathbf{x}^{\mathrm{F}}_j 1\Big\}_{j=1}^{J} \subseteq \mathcal{S}_{\texttt{opt}} \tag{18}$$

Proof. We prove this step by contradiction. Assume there exists an optimal tokeniser with vocabulary $\mathcal{S}_{\text{✗}}$ which does not include $t > 0$ of the tokens above. Now, choose an arbitrary set of t tokens in this vocabulary which are not of the form above, and replace them with the missing tokens in this set. We denote this new tokeniser's vocabulary $\mathcal{S}_{\text{✓}}$. Note that the strings in $\mathcal{D}_1$ with these missing tokens were represented with at least 2 symbols under $\mathcal{S}_{\text{✗}}$, but with a single token under $\mathcal{S}_{\text{✓}}$, i.e.,:

$$\mathfrak{G}_\ell(\texttt{tok}_{\diamondsuit}[\mathcal{S}_{\text{✗}}],\mathcal{D}_1) \geq (4J+t)f, \qquad \mathfrak{G}_\ell(\texttt{tok}_{\diamondsuit}[\mathcal{S}_{\text{✓}}],\mathcal{D}_1) = 4Jf \tag{19}$$

Further, note that under $\mathcal{S}_{\text{✓}}$, we have that strings in dataset $\mathcal{D}_2$ are compressed to at most two symbols, while strings in $\mathcal{D}_3$ and $\mathcal{D}_4$ are compressed to at most three symbols:

$$\forall \mathbf{c} \in \mathcal{D}_2 : \mathfrak{G}_\ell(\texttt{tok}_{\diamondsuit}[\mathcal{S}_{\text{✓}}],\mathbf{c}) \leq 2, \quad \forall \mathbf{c} \in \mathcal{D}_3 \cup \mathcal{D}_4 : \mathfrak{G}_\ell(\texttt{tok}_{\diamondsuit}[\mathcal{S}_{\text{✓}}],\mathbf{c}) \leq 3 \tag{20}$$

To improve on this compressed length, $\mathcal{S}_{\text{✗}}$ must, thus, compress strings in $\mathcal{D}_2$ to a single symbol, or strings in $\mathcal{D}_3$ and $\mathcal{D}_4$ to one or two symbols. Notably, this can only be done if the noncompliant tokens in $\mathcal{S}_{\text{✗}}$ contain 101-strings. This is because, to compress a string in $\mathcal{D}_2$ to a single symbol, the full character-string must become a token, and $\mathcal{D}_2$ only includes 101-strings. Further, under $\mathcal{S}_{\text{✓}}$, strings in $\mathcal{D}_3$ and $\mathcal{D}_4$ are already compressed to at least $\langle 1\mathbf{x}^{\mathrm{T}}_j, 1, \mathbf{x}^{\mathrm{T}}_j 1\rangle$. To further compress them, tokeniser $\mathcal{S}_{\text{✗}}$ must include tokens which cross the "middle" of this character-string, which would make this tokens at least have a 101 prefix or suffix. We consider the best case scenario, which is if they are exactly 101-strings, as any longer string will be at most as frequent as it.

As discussed above, however, each 101-string appears at most: f' times in $\mathcal{D}_2$, f'' times in $\mathcal{D}_3$, 3 times in $\mathcal{D}_4$. This gives us a best case scenario—in which all the strings in which a new token appears are compressed to a single symbol—where:

$$\mathfrak{G}_\ell(\texttt{tok}_{\diamondsuit}[\mathcal{S}_{\text{✓}}],\mathcal{D}_2 \cup \mathcal{D}_3 \cup \mathcal{D}_4) - \mathfrak{G}_\ell(\texttt{tok}_{\diamondsuit}[\mathcal{S}_{\text{✗}}],\mathcal{D}_2 \cup \mathcal{D}_3 \cup \mathcal{D}_4) \leq t(f' + 2(f'' + 3)) \tag{21}$$

As the difference in Eq. (19) is of at least tf tokens, we put these together:

$$\mathfrak{G}_\ell(\texttt{tok}_\diamond[\mathcal{S}_{\checkmark}], \mathcal{D}) - \mathfrak{G}_\ell(\texttt{tok}_\diamond[\mathcal{S}_{\text{✗}}], \mathcal{D}) \leq t(f' + 2(f'' + 3)) - tf \tag{22}$$

As $f > f' + 2(f'' + 3)$, this difference is smaller than zero, implying that $\mathcal{S}_{\checkmark}$ improves on $\mathcal{S}_{\text{✗}}$. This shows a contradiction, which completes our proof. □

LemmaProofStep 2. (Step ②). *An optimal tokeniser must include all tokens of the form* $1\mathrm{x}_j^{\mathrm{T}}, \mathrm{x}_j^{\mathrm{T}}1, 1\mathrm{x}_j^{\mathrm{F}}, \mathrm{x}_j^{\mathrm{F}}1$, *and further only tokens of the form* $1\mathrm{x}_j^{\mathrm{T}}1, 1\mathrm{x}_j^{\mathrm{F}}1$, *i.e.,:*

$$\left\{1\mathrm{x}_j^{\mathrm{T}}, \mathrm{x}_j^{\mathrm{T}}1, 1\mathrm{x}_j^{\mathrm{F}}, \mathrm{x}_j^{\mathrm{F}}1\right\}_{j=1}^{J} \subseteq \mathcal{S}_{\text{opt}} \quad \text{and} \quad \mathcal{S}_{\text{opt}} \subset \left\{1\mathrm{x}_j^{\mathrm{T}}, \mathrm{x}_j^{\mathrm{T}}1, 1\mathrm{x}_j^{\mathrm{F}}, \mathrm{x}_j^{\mathrm{F}}1, 1\mathrm{x}_j^{\mathrm{T}}1, 1\mathrm{x}_j^{\mathrm{F}}1\right\}_{j=1}^{J} \tag{23}$$

Proof. As before, we prove this step by contradiction. Given step ① above, we know an optimal tokeniser includes all tokens $1\mathrm{x}_j^{\mathrm{T}}, \mathrm{x}_j^{\mathrm{T}}1, 1\mathrm{x}_j^{\mathrm{F}}, \mathrm{x}_j^{\mathrm{F}}1$. Now, assume there exists an optimal tokeniser with vocabulary $\mathcal{S}_{\text{✗}}$ with $t > 0$ tokens which are not of the form $1\mathrm{x}_j^{\mathrm{T}}, \mathrm{x}_j^{\mathrm{T}}1, 1\mathrm{x}_j^{\mathrm{F}}, \mathrm{x}_j^{\mathrm{F}}1$ or $1\mathrm{x}_j^{\mathrm{T}}1, 1\mathrm{x}_j^{\mathrm{F}}1$; we will call these tokens non-compliant here. Choose an arbitrary set of t unused compliant tokens—i.e., with form $1\mathrm{x}_j^{\mathrm{T}}1, 1\mathrm{x}_j^{\mathrm{F}}1$—to replace the non-compliant tokens with, forming a new tokeniser's vocabulary $\mathcal{S}_{\checkmark}$. Both these vocabularies compress strings in $\mathcal{D}_1$ equally:

$$\mathfrak{G}_\ell(\texttt{tok}_\diamond[\mathcal{S}_{\text{✗}}], \mathcal{D}_1) = 4Jf, \qquad \mathfrak{G}_\ell(\texttt{tok}_\diamond[\mathcal{S}_{\checkmark}], \mathcal{D}_1) = 4Jf \tag{24}$$

For strings in $\mathcal{D}_2$: if the entire string is in the vocabulary, it is encoded as a single token; else, it is represented with two symbols. Under $\mathcal{S}_{\checkmark}$, there are J tokens covering strings in $\mathcal{D}_2$. Under $\mathcal{S}_{\text{✗}}$, there are only $(J - t)$ tokens covering strings in $\mathcal{D}_2$. This implies:

$$\mathfrak{G}_\ell(\texttt{tok}_\diamond[\mathcal{S}_{\text{✗}}], \mathcal{D}_2) = (3J + t)f', \qquad \mathfrak{G}_\ell(\texttt{tok}_\diamond[\mathcal{S}_{\checkmark}], \mathcal{D}_2) = 3Jf' \tag{25}$$

Finally, for strings in $\mathcal{D}_3$ and $\mathcal{D}_4$, a similar argument to the previous step applies: (i) only tokens containing 101-strings can compress these datasets; (ii) each 101-string appears at most $f'' + 3$ times in them; (iii) each 101-string will lead to at most two symbols being saved. As $\mathcal{S}_{\text{✗}}$ differs from $\mathcal{S}_{\checkmark}$ in t tokens, we get that it will improve on it by at most:

$$\mathfrak{G}_\ell(\texttt{tok}_\diamond[\mathcal{S}_{\checkmark}], \mathcal{D}_3 \cup \mathcal{D}_4) - \mathfrak{G}_\ell(\texttt{tok}_\diamond[\mathcal{S}_{\text{✗}}], \mathcal{D}_3 \cup \mathcal{D}_4) \leq 2t(f'' + 3) \tag{26}$$

Summing together the compression on all datasets, we get their difference is bounded by:

$$\mathfrak{G}_\ell(\texttt{tok}_\diamond[\mathcal{S}_{\checkmark}], \mathcal{D}) - \mathfrak{G}_\ell(\texttt{tok}_\diamond[\mathcal{S}_{\text{✗}}], \mathcal{D}) \leq 2t(f'' + 3) - tf' \tag{27}$$

As $f' > 2(f'' + 3)$, this difference is smaller than zero, implying that $\mathcal{S}_{\checkmark}$ improves on $\mathcal{S}_{\text{✗}}$. This shows a contradiction, which completes our proof. □

LemmaProofStep 3. (Step ③). *An optimal tokeniser must be* `sat`*-compliant: it must contain all tokens of the form* $1\mathrm{x}_j^{\mathrm{T}}, \mathrm{x}_j^{\mathrm{T}}1, 1\mathrm{x}_j^{\mathrm{F}}, \mathrm{x}_j^{\mathrm{F}}1$ *and it must contain either* $1\mathrm{x}_j^{\mathrm{T}}1$ *or* $1\mathrm{x}_j^{\mathrm{F}}1$ *for each* $1 \leq j \leq J$.

Proof. As before, we prove this step by contradiction. Given step ① above, we know an optimal tokeniser includes all tokens $1\mathrm{x}_j^{\mathrm{T}}, \mathrm{x}_j^{\mathrm{T}}1, 1\mathrm{x}_j^{\mathrm{F}}, \mathrm{x}_j^{\mathrm{F}}1$. Further, given step ② above, we know its other tokens all have form $1\mathrm{x}_j^{\mathrm{T}}1, 1\mathrm{x}_j^{\mathrm{F}}1$. Now, assume there exists an optimal tokeniser with vocabulary $\mathcal{S}_{\text{✗}}$ which includes both $1\mathrm{x}_j^{\mathrm{T}}1$ and $1\mathrm{x}_j^{\mathrm{F}}1$ for $t > 0$ variables, and thus neither of those two for $t > 0$ other variables. Then, define $\mathcal{S}_{\checkmark}$ as a vocabulary where the $1\mathrm{x}_j^{\mathrm{F}}1$ token of all t doubly assigned variables are replaced with the $1\mathrm{x}_j^{\mathrm{T}}1$ token of all non-assigned variables. Note that $\mathcal{S}_{\checkmark}$ is `sat`-compliant. These two tokenisers achieve the same compression on $\mathcal{D}_1$ and $\mathcal{D}_2$:

$$\mathfrak{G}_\ell(\texttt{tok}_\diamond[\mathcal{S}_{\text{✗}}], \mathcal{D}_1 \cup \mathcal{D}_2) = 4Jf + 3Jf', \qquad \mathfrak{G}_\ell(\texttt{tok}_\diamond[\mathcal{S}_{\checkmark}], \mathcal{D}_1 \cup \mathcal{D}_2) = 4Jf + 3Jf' \tag{28}$$

The tokeniser with vocabulary $\mathcal{S}_{\checkmark}$ will then compress each string in $\mathcal{D}_3$ to 2 symbols, while $\mathcal{S}_{\text{✗}}$ will only compress the t strings $1\mathrm{x}_j^{\mathrm{T}}1\mathrm{x}_j^{\mathrm{F}}1$ with unassigned variables to 3 symbols. This will lead to a total compression of:

$$\mathfrak{G}_\ell(\texttt{tok}_\diamond[\mathcal{S}_{\text{✗}}], \mathcal{D}_3) = (2J + t)f'', \qquad \mathfrak{G}_\ell(\texttt{tok}_\diamond[\mathcal{S}_{\checkmark}], \mathcal{D}_3) = 2Jf'' \tag{29}$$

Finally, the t doubly assigned tokens of the form $1\mathrm{x}_j^{\mathrm{F}}1$ (which $\mathcal{S}_{\checkmark}$ does not contain) appear at most three times in $\mathcal{D}_4$ and will lead to at most one symbol being saved, leading to a bound:

$$\mathfrak{G}_\ell(\texttt{tok}_\diamond[\mathcal{S}_{\text{✗}}], \mathcal{D}_4) - \mathfrak{G}_\ell(\texttt{tok}_\diamond[\mathcal{S}_{\checkmark}], \mathcal{D}_4) \leq 3t \tag{30}$$

Putting these compressed lengths together, we get:

$$\mathfrak{G}_\ell(\texttt{tok}_\Phi[\mathcal{S}_\text{✗}],\mathcal{D}) - \mathfrak{G}_\ell(\texttt{tok}_\Phi[\mathcal{S}_\text{✓}],\mathcal{D}) \leq 3t - tf'' \tag{31}$$

As $f'' > 3$, this difference is smaller than zero, implying that $\mathcal{S}_\text{✓}$ improves on $\mathcal{S}_\text{✗}$. This shows a contradiction, which completes our proof. □

LemmaProofStep 4. (Step ④). *If an optimal tokeniser achieves a compressed length of at least* $329J + 3I - \gamma$*, the original* 3-OCC-MAX2SAT *instance is satisfiable, i.e.,:*

$$\Big(\mathfrak{G}_\ell(\texttt{tok}_\Phi[\mathcal{S}_\text{opt}],\mathcal{D}) \leq 329J + 3I - \gamma\Big) \implies \text{3OM2S}(\mathcal{X},\mathcal{C},\gamma) \tag{32}$$

Proof. Given steps ① to ③, we know that an optimal tokeniser will be sat-compliant. We will now denote this optimal tokeniser's vocabulary as $\mathcal{S}_\text{opt}$ and use Eq. (15) to extract a 3-OCC-MAX2SAT assignment $\mathcal{X}^\star = g(\mathcal{S}_\text{opt})$ which corresponds to this tokeniser's vocabulary. From the previous proof steps we see that any sat-compliant tokeniser achieves the following compressed length in $\mathcal{D}_1$, $\mathcal{D}_2$, and $\mathcal{D}_3$:

$$\mathfrak{G}_\ell(\texttt{tok}_\Phi[\mathcal{S}_\text{opt}],\mathcal{D}_1 \cup \mathcal{D}_2 \cup \mathcal{D}_3) = 4Jf + 3Jf' + 2Jf'' = 329J \tag{33}$$

Now, note that a character-string $1L_i^1 1 L_i^2 1$ in $\mathcal{D}_4$ will be: compressed to two symbols if at least one of the tokens $1L_i^1 1$ or $1L_i^2 1$ exists, or compressed to three symbols if neither exists. Equivalently, a clause $L_i^1 \vee L_i^2$ in 3-OCC-MAX2SAT is: satisfied, if either L_i^1 or L_i^2 evaluates to true; not satisfied, if both evaluate to false. Given our construction of function g above, one of 3-OCC-MAX2SAT's clauses will be satisfied if and only if its corresponding string in $\mathcal{D}_4$ is compressed to two symbols. We can thus state that:

$$\Big(\mathfrak{G}_\ell(\texttt{tok}_\Phi[\mathcal{S}_\text{opt}],\mathcal{D}_4) = 3I - \gamma^\star\Big) \iff \left(\sum_{i=1}^{I} \mathbb{1}_{\mathcal{X}^\star}\{L_i^1 \vee L_i^2\} = \gamma^\star\right) \tag{34}$$

Given the construction of δ as $329J + 3I - \gamma$, we conclude that a sat-compliant tokeniser which compresses the full dataset to at least that size can be mapped to a 3-OCC-MAX2SAT assignment which satisfies at least γ clauses. This concludes the proof. □

C Proof of Lemma 3

Lemma 3. *The direct binary tokenisation gap problem is* NP*-hard.*

Proof. For this proof, we rely on a result by Berman and Karpinski (1998; 1999) that, for specific instances of 3-OCC-MAX2SAT with $I = 2016n$ clauses, it is NP-hard to distinguish whether at least $(2012 - \varepsilon)n$ or at most $(2011 + \varepsilon)n$ of those are satisfiable, for any $\varepsilon > 0$. We will denote this 3-OCC-MAX2SAT gap problem as $\text{3OM2S}(\mathcal{X},\mathcal{C},(\gamma^-,\gamma^+))$, with $\gamma^- = (2011 + \varepsilon)n$ and $\gamma^+ = (2012 - \varepsilon)n$. We can now prove the NP-hardness of the direct binary tokenisation gap problem by reducing 3-OCC-MAX2SAT's gap problem to it. To this end, we rely on a reduction identical to $\text{R1}(\mathcal{X},\mathcal{C},\gamma)$, but where we define:

$$\begin{aligned} \delta^- &= 329J + 3I - \gamma^- & \delta^+ &= 329J + 3I - \gamma^+ \\ &= 329J + 3I - \frac{2011+\varepsilon}{2016} I & &= 329J + 3I - \frac{2012-\varepsilon}{2016} I \end{aligned} \tag{35}$$

Lemmas 1 and 2 trivially show the validity of this reduction:

$$\text{3OM2S}(\mathcal{X},\mathcal{C},(\gamma^-,\gamma^+)) \iff \text{Tok}_\Phi^2(\mathcal{D},K,(\delta^-,\delta^+)) \tag{36}$$

which holds since $\text{3OM2S}(\mathcal{X},\mathcal{C},\gamma^+) \iff \text{Tok}_\Phi^2(\mathcal{D},K,\delta^+)$ and the same for γ^- and δ^-. It is therefore NP-hard to distinguish whether a dataset can be compressed to at most $329J + 3I - \frac{2012-\varepsilon}{2016} I$ symbols, or if at least $329J + 3I - \frac{2011+\varepsilon}{2016} I$ symbols remain (with an allowed vocabulary size $K = 5J$). Because each variable occurs exactly three times in 3-OCC-MAX2SAT, we have that $\frac{3}{2}J = I$. We now compute the maximum achievable compression ratio:

$$\frac{\delta^-}{\delta^+} = \frac{329J + 3I - \frac{2011+\varepsilon}{2016} I}{329J + 3I - \frac{2012-\varepsilon}{2016} I} \tag{37a}$$

$$= \frac{667 - \frac{6033+3\varepsilon}{2016}}{667 - \frac{6036-\varepsilon}{2016}} \tag{37b}$$

$$= \frac{1338639 - 3\varepsilon}{1338636 + 3\varepsilon} \tag{37c}$$

$$= \frac{446213 - \varepsilon}{446212 + \varepsilon} \tag{37d}$$

Thus, direct binary tokenisation cannot be approximated in polynomial time with an approximation ratio better than $\frac{446213}{446212} > 1.000002$ unless $\mathsf{P} = \mathsf{NP}$. □

D Proof of Lemma 5

We first need another lemma in preparation for the actual proof. Note that in a merge sequence, simply merging $\bigcirc_{j=1}^{2J-1}[\langle 0^j, 0\rangle]$ does not compress all targets in $\{0^j \mid 1 \leq j \leq 2J\}$ to one symbol, because the first merge creates lots of 00 tokens. Thus, we need to describe a more unwieldy merge sequence to achieve this with the same number of merges.

Lemma 4. *Given n targets $\{0, \ldots, 0^J\}$ OMS/OPE would require exactly $J - 1$ merges in order for all of them to be merged into a single token.*

Proof. We establish the result in two steps: first a lower bound, then a matching constructive upper bound.

LemmaProofStep 1. (Step ①). *Each merge can reduce at most one multi-token target to a single token.*

Proof. The target set contains J distinct values, one of which is the base symbol 0. Whenever a target becomes a single token through a merge, that merge must combine exactly two tokens whose sum equals that specific target. Because all targets are distinct, this sum cannot simultaneously equal any other target. Hence, a single merge can complete at most one target. □

From Step ①, it follows that at least $J - 1$ merges are required to reduce all targets to single tokens.

LemmaProofStep 2. (Step ②). *There exists an explicit merge sequence that reduces all targets to single tokens in exactly $J - 1$ merges.*

For the matching upper bound, consider the following explicit merge sequence on the input $\{0, \ldots, 0^J\}$. Let k be such that $2^k \leq J < 2^{k+1}$.

1. **Binary stage.** First perform k merges to construct single tokens for each power of two up to 2^k
2. **Extension stage.** At each next step, apply the rightmost merge on the smallest non-fully merged value.[9]

After these initial k merges, every target can be expressed in binary as a sum of powers of two, i.e., as a sequence of tokens representing distinct powers of two. Moreover, each of these k merges produces exactly one additional single-token target.

LemmaProofStep 3. (Step ③). *In the merge sequence from Step ②, the smallest unmerged target always consists of exactly two tokens.*

Proof. Assume for contradiction that at some point the smallest unmerged target r consists of more than two tokens.

After the binary stage (Step ②.1), every target can be expressed in a canonical way: each target is written as a sum of tokens, where every token is either (1) a power of two (created in the binary stage),

[9] As we will see in the proof, the smallest non-fully merged value always consists of only two symbols, such that the "rightmost" tiebreaker is not necessary.

or (2) a single leftover "tail" that is smaller than the power of two immediately before it. This gives a unique decomposition for every target: no target can be represented by two different sequences of such tokens.

Write the last two tokens of r as $2^{c_{m-1}}$ and c_m with $c_m < 2^{c_{m-1}}$. Consider the smaller target

$$r' = 2^{c_{m-1}} + c_m. \tag{38}$$

Because $r' < r$ and the merge sequence always collapses the smallest unmerged target first, the pair $\langle 2^{c_{m-1}}, c_m \rangle$ must already have been merged into a single token when r' was processed. By uniqueness of decomposition, this merge must occur in exactly the same way inside r.

Hence r cannot still contain the two separate tokens $2^{c_{m-1}}$ and c_m, contradicting the assumption that r has more than two tokens.

Therefore, the smallest unmerged target must always consist of exactly two tokens.

□

Step ① shows that no fewer than $J-1$ merges are required. Step ② constructs an explicit merge sequence, and Step ③ ensures that in this sequence each merge produces exactly one additional single-token target. Together, these steps establish that OMS/OPE requires precisely $J-1$ merges.

□

With this, we can now prove the lemma.

Lemma 5. *If a* 3-OCC-MAX2SAT *instance is satisfiable, then the* B-2-TOK *instance output by Reduction 2 is also satisfiable. Formally:* $\mathrm{3OM2S}(\mathcal{X}, \mathcal{C}, \gamma) \implies \mathrm{Tok}^2_{\uparrow}(\mathrm{R1}(\mathcal{X}, \mathcal{C}, \gamma))$.

Proof. Assume this $(\mathcal{X}, \mathcal{C}, \gamma)$ instance of the 3-OCC-MAX2SAT problem is satisfiable, i.e., that $\mathrm{3OM2S}(\mathcal{X}, \mathcal{C}, \gamma)$ is true. We must prove that in this case, $\mathrm{Tok}^2_{\uparrow}(\mathrm{R1}(\mathcal{X}, \mathcal{C}, \gamma))$ is also true. We define the following list of merges which, as shown in Lemma 4, compress every target of type $\mathbf{x}^{\mathsf{T}}_j, \mathbf{x}^{\mathsf{F}}_j$ to a single token

$$\mathbf{m}_1 = \langle 1, 1 \rangle \circ \bigcirc_{j=1}^{\lfloor log(2J-1) \rfloor} [\langle 0^{2^i}, 0^{2^i} \rangle] \circ \bigcirc_{j=1}^{\lfloor log(2J-1) \rfloor} \bigcirc_{j'=1}^{2^j-1} [\langle 0^{2^j}, 0^{j'} \rangle] \tag{39a}$$

Note that the merge $\langle 1, 1 \rangle := \otimes$ is independent of the merges on 0 and could thus be placed at any point in the sequence.

We also define the following lists of merges, which will be included in any satisfying solution to the tokenisation problem

$$\mathbf{m}_2 = \bigcirc_{j=1}^{J} [\langle 11, \mathbf{x}^{\mathsf{F}}_j \rangle, \langle \mathbf{x}^{\mathsf{T}}_j, 11 \rangle] \tag{40}$$

$$\mathbf{m}_4 = \bigcirc_{j=1}^{J} [\langle \mathbf{x}^{\mathsf{F}}_j, 1 \rangle, \langle 1, \mathbf{x}^{\mathsf{T}}_j \rangle] \tag{41}$$

$$\mathbf{m}_6 = \bigcirc_{j=1}^{J} [\langle 1, \mathbf{x}^{\mathsf{F}}_j \rangle, \langle \mathbf{x}^{\mathsf{T}}_j, 1 \rangle] \tag{42}$$

Now, let $\mathcal{X}^\star = \{x^\star_j\}_{j=1}^J$ be any satisfying solution to this $(\mathcal{X}, \mathcal{C}, \gamma)$ instance of the 3-OCC-MAX2SAT problem. We define the following instance-specific merges

$$\mathbf{m}_3 = \bigcirc_{j=1}^{J} \begin{bmatrix} \langle 1, \mathbf{x}^{\mathsf{T}}_j 11 \rangle & \texttt{if } x^\star_j = \mathsf{T} \\ \langle 11 \mathbf{x}^{\mathsf{F}}_j, 1 \rangle & \texttt{else} \end{bmatrix}, \qquad \mathbf{m}_5 = \bigcirc_{j=1}^{J} \begin{bmatrix} \langle 1 \mathbf{x}^{\mathsf{T}}_j, 1 \rangle & \texttt{if } x^\star_j = \mathsf{T} \\ \langle 1, \mathbf{x}^{\mathsf{F}}_j 1 \rangle & \texttt{else} \end{bmatrix} \tag{43}$$

In words, we create merges $\langle 1, \mathbf{x}^{\mathsf{T}}_j 11 \rangle$ and $\langle 1 \mathbf{x}^{\mathsf{T}}_j, 1 \rangle$ if $x^\star_j$ is true, or $\langle 11 \mathbf{x}^{\mathsf{F}}_j, 1 \rangle$ and $\langle 1, \mathbf{x}^{\mathsf{F}}_j 1 \rangle$ if $x^\star_j$ is false. We then create a merge sequence by concatenating these lists in order:

$$\mathbf{m} = \mathbf{m}_1 \circ \mathbf{m}_2 \circ \mathbf{m}_3 \circ \mathbf{m}_4 \circ \mathbf{m}_5 \circ \mathbf{m}_6 \tag{44}$$

This gives us a total of $|\mathbf{m}| = K = 10J$ merges. Now we just need to count the symbols output by this solution to see if the bound is satisfied.

By applying the merges $\mathbf{m}$, each string in $\mathcal{D}_1$ will be compressed into a single subword, obtaining

$$\sum_{\mathbf{c} \in (\bigcup_{-=1}^{f} \mathcal{D}_1)} |\mathtt{tok}_{\uparrow}[\mathbf{m}](\mathbf{c})| = 108 f J \tag{45}$$

Table 1: Performance of merges on strings in $\mathcal{D}_5$, adapted from Whittington et al. (2025). The dot symbol · denotes the string not changing under the given merge.

Assignment	Condition	$\mathbf{c}$	$\texttt{tok}_\uparrow[\mathbf{m}_1](\mathbf{c})$	$\texttt{tok}_\uparrow[\mathbf{m}_1 \circ \mathbf{m}_2](\mathbf{c})$	$\texttt{tok}_\uparrow[\mathbf{m}_1 \circ \mathbf{m}_2 \circ \mathbf{m}_3](\mathbf{c})$	$\texttt{tok}_\uparrow[\mathbf{m}_1 \circ \cdots \circ \mathbf{m}_4](\mathbf{c})$	$\texttt{tok}_\uparrow[\mathbf{m}_1 \circ \cdots \circ \mathbf{m}_5](\mathbf{c})$	$\lvert\texttt{tok}_\uparrow[\mathbf{m}](\mathbf{c})\rvert$
$L_i^1 = X_j$ and $L_i^2 = \neg X_{j'}$	$x_j^\star = \texttt{T} \wedge x_{j'}^\star = \texttt{T}$	$\langle 1,0,\ldots,0,1,\underbrace{0,\ldots,0}_{2j'},1\rangle$	$\langle 1, \mathbf{x}_j^\texttt{T}, 1, \mathbf{x}_{j'}^\texttt{F}, 1\rangle$	·	·	$\langle 1\mathbf{x}_j^\texttt{T}, 1, \mathbf{x}_{j'}^\texttt{F}, 1\rangle$	$\langle 1\mathbf{x}_j^\texttt{T}1, \mathbf{x}_{j'}^\texttt{F}, 1\rangle$	2
	$x_j^\star = \texttt{F} \wedge x_{j'}^\star = \texttt{T}$			·	·		$\langle 1\mathbf{x}_j^\texttt{F}, 1, \mathbf{x}_{j'}^\texttt{F}, 1\rangle$	3
	$x_j^\star = \texttt{T} \wedge x_{j'}^\star = \texttt{F}$			·	·		$\langle 1\mathbf{x}_j^\texttt{T}1, \mathbf{x}_{j'}^\texttt{F}, 1\rangle$	2
	$x_j^\star = \texttt{F} \wedge x_{j'}^\star = \texttt{F}$			·	·		$\langle 1\mathbf{x}_j^\texttt{F}, 1\mathbf{x}_{j'}^\texttt{F}, 1\rangle$	2
$L_i^1 = \neg X_j$ and $L_i^2 = X_{j'}$	$x_j^\star = \texttt{T} \wedge x_{j'}^\star = \texttt{T}$	$\langle 1,\underbrace{0,\ldots,0}_{2j},1,\underbrace{0,\ldots,0}_{2j},1\rangle$	$\langle 1, \mathbf{x}_j^\texttt{T}, 1, \mathbf{x}_{j'}^\texttt{F}, 1\rangle$	·	·	$\langle 1\mathbf{x}_j^\texttt{T}, 1, \mathbf{x}_{j'}^\texttt{F}1\rangle$	$\langle 1\mathbf{x}_j^\texttt{T}, 1, \mathbf{x}_{j'}^\texttt{F}1\rangle$	2
	$x_j^\star = \texttt{F} \wedge x_{j'}^\star = \texttt{T}$			·	·		$\langle 1\mathbf{x}_j^\texttt{F}, 1, \mathbf{x}_{j'}^\texttt{F}1\rangle$	2
	$x_j^\star = \texttt{T} \wedge x_{j'}^\star = \texttt{F}$			·	·		$\langle 1\mathbf{x}_j^\texttt{T}, 1, \mathbf{x}_{j'}^\texttt{F}1\rangle$	3
	$x_j^\star = \texttt{F} \wedge x_{j'}^\star = \texttt{F}$			·	·		$\langle 1\mathbf{x}_j^\texttt{F}, 1, \mathbf{x}_{j'}^\texttt{F}1\rangle$	2
$L_i^1 = \neg X_j$ and $L_i^2 = \neg X_{j'}$	$x_j^\star = \texttt{T} \wedge x_{j'}^\star = \texttt{T}$	$\langle 1,\underbrace{0,\ldots,0}_{2j},1,\underbrace{0,\ldots,0}_{2j'},1\rangle$	$\langle 11, \mathbf{x}_j^\texttt{F}, 1, \mathbf{x}_{j'}^\texttt{F}, 1\rangle$	·	$\langle 11\mathbf{x}_j^\texttt{F}1, \mathbf{x}_{j'}^\texttt{F}, 1\rangle$	·	·	3
	$x_j^\star = \texttt{F} \wedge x_{j'}^\star = \texttt{T}$			$\langle 11\mathbf{x}_j^\texttt{F}1, \mathbf{x}_{j'}^\texttt{F}, 1\rangle$	$\langle 11\mathbf{x}_j^\texttt{F}1, \mathbf{x}_{j'}^\texttt{F}, 1\rangle$	$\langle 11\mathbf{x}_j^\texttt{F}, 1, \mathbf{x}_{j'}^\texttt{F}, 1\rangle$	·	2
	$x_j^\star = \texttt{T} \wedge x_{j'}^\star = \texttt{F}$			·	$\langle 11\mathbf{x}_j^\texttt{F}, 1, \mathbf{x}_{j'}^\texttt{F}, 1\rangle$		$\langle 11\mathbf{x}_j^\texttt{F}, 1\mathbf{x}_{j'}^\texttt{F}, 1\rangle$	2
	$x_j^\star = \texttt{F} \wedge x_{j'}^\star = \texttt{F}$			$\langle 11\mathbf{x}_j^\texttt{F}1, \mathbf{x}_{j'}^\texttt{F}, 1\rangle$	$\langle 11\mathbf{x}_j^\texttt{F}1, \mathbf{x}_{j'}^\texttt{F}, 1\rangle$		·	2
$L_i^1 = X_j$ and $L_i^2 = X_{j'}$	$x_j^\star = \texttt{T} \wedge x_{j'}^\star = \texttt{T}$	$\langle 1,\underbrace{0,\ldots,0}_{2j-1},1,\underbrace{0,\ldots,0}_{2j'-1},1\rangle$	$\langle 1, \mathbf{x}_j^\texttt{T}, 1, \mathbf{x}_{j'}^\texttt{T}, 11\rangle$	$\langle 1, \mathbf{x}_j^\texttt{T}, 1\mathbf{x}_{j'}^\texttt{T}, 11\rangle$	$\langle 1\mathbf{x}_j^\texttt{T}, 1\mathbf{x}_{j'}^\texttt{T}, 11\rangle$	$\langle 1\mathbf{x}_j^\texttt{T}, 1\mathbf{x}_{j'}^\texttt{T}, 11\rangle$	·	2
	$x_j^\star = \texttt{F} \wedge x_{j'}^\star = \texttt{T}$			·	$\langle 1\mathbf{x}_j^\texttt{T}, 1, \mathbf{x}_{j'}^\texttt{T}, 11\rangle$			2
	$x_j^\star = \texttt{T} \wedge x_{j'}^\star = \texttt{F}$							2
	$x_j^\star = \texttt{F} \wedge x_{j'}^\star = \texttt{F}$						·	3

For each pair of strings $1\mathbf{x}_j^\texttt{T}1$ and $1\mathbf{x}_j^\texttt{F}1$ in $\mathcal{D}_2$, one is compressed into a single subword while the other is only compressed to two subwords—the one with $\mathbf{x}_j^\texttt{T}$ is compressed to a single symbol if $x_j^\star = \texttt{T}$ and the one with $\mathbf{x}_j^\texttt{F}$ otherwise. The same is true for each pair of strings $1\mathbf{x}_j^\texttt{T}11$ and $11\mathbf{x}_j^\texttt{F}1$, also in $\mathcal{D}_2$. We thus have that, for each variable X_j, the strings in $\mathcal{D}_2$ will occupy a total of $(1+2+1+2)J$ characters, and:

$$\sum_{\mathbf{c} \in (\bigcup_{j=1}^{f} \mathcal{D}_2)} \lvert\texttt{tok}_\uparrow[\mathbf{m}](\mathbf{c})\rvert = 6f'J \tag{46}$$

Similarly, each string in $\mathcal{D}_3$ and $\mathcal{D}_4$ will be compressed into only 2 symbols after this tokeniser is applied to it. We thus have:

$$\sum_{\mathbf{c} \in (\bigcup_{j=1}^{f''} \mathcal{D}_3)} \lvert\texttt{tok}_\uparrow[\mathbf{m}](\mathbf{c})\rvert = 4f''J, \qquad \sum_{\mathbf{c} \in (\bigcup_{j=1}^{f'''} \mathcal{D}_4)} \lvert\texttt{tok}_\uparrow[\mathbf{m}](\mathbf{c})\rvert = 4f'''J \tag{47}$$

Finally, we have the strings in $\mathcal{D}_5$. These strings are constructed such that they will be compressed into 2 symbols if either L_i^1 or L_i^2 evaluates to $\texttt{T}$, and kept with 3 symbols otherwise; see Tab. 1 for a detailed simulation of why this is the case.

We thus have:

$$\sum_{\mathbf{c} \in \mathcal{D}_5} \lvert\texttt{tok}_\uparrow[\mathbf{m}](\mathbf{c})\rvert = \sum_{i=1}^{I} \left(3 - 1\,\mathbb{1}\left\{ \begin{array}{c} L_i^1 = X_j \quad \texttt{and} \ \langle 1, \mathbf{x}_j^\texttt{T}11\rangle, \langle 1\mathbf{x}_j^\texttt{T}, 1\rangle \in \mathbf{m} \\ \text{or} \\ L_i^1 = \neg X_j \ \texttt{and} \ \langle 11\mathbf{x}_j^\texttt{F}, 1\rangle, \langle 1, \mathbf{x}_j^\texttt{F}1\rangle \in \mathbf{m} \\ \text{or} \\ L_i^2 = X_{j'} \quad \texttt{and} \ \langle 1, \mathbf{x}_{j'}^\texttt{T}11\rangle, \langle 1\mathbf{x}_{j'}^\texttt{T}, 1\rangle \in \mathbf{m} \\ \text{or} \\ L_i^2 = \neg X_{j'} \ \texttt{and} \ \langle 11\mathbf{x}_{j'}^\texttt{F}, 1\rangle, \langle 1, \mathbf{x}_{j'}^\texttt{F}1\rangle \in \mathbf{m} \end{array} \right\} \right) \tag{48a}$$

$$= 3I - \sum_{i=1}^{I} \mathbb{1}_{\mathcal{X}^\star}\{L_i^1 \vee L_i^2\} \tag{48b}$$

$$\leq 3I - \gamma \tag{48c}$$

where, by construction, we have a merge in our sequence (e.g., $\langle 1, \mathbf{x}_j^\texttt{T}11\rangle$ or $\langle 11\mathbf{x}_j^\texttt{F}, 1\rangle$) if and only if its value is in a satisfying assignment (e.g., $x_j^\star = \texttt{T}$ or $x_j^\star = \texttt{F}$ respectively). Summing together the lengths in Eqs. (45) to (48), we get that:

$$\sum_{\mathbf{c} \in \mathcal{D}} \lvert\texttt{tok}_\uparrow[\mathbf{m}](\mathbf{c})\rvert \leq \delta = (108f + 6f' + 4f'' + 4f''')\,J + 3\,I - \gamma \tag{49}$$

which concludes the proof. □

E Proof of Lemma 6

We again start with some useful definitions and redefine compliant tokenisers.

We define a **sat-compliant (direct) tokeniser** to be any tokeniser which: (i) contains all tokens of the form $11, 1\mathbf{x}_j^{\mathtt{T}}, \mathbf{x}_j^{\mathtt{F}}, 1\mathbf{x}_j^{\mathtt{T}}, \mathbf{x}_j^{\mathtt{T}}1, 1\mathbf{x}_j^{\mathtt{F}}, \mathbf{x}_j^{\mathtt{F}}1, \mathbf{x}_j^{\mathtt{T}}11, 11\mathbf{x}_j^{\mathtt{F}}$; and (ii) contains either $1\mathbf{x}_j^{\mathtt{T}}1, 1\mathbf{x}_j^{\mathtt{T}}11$ or $1\mathbf{x}_j^{\mathtt{F}}1, 11\mathbf{x}_j^{\mathtt{F}}1$ for each $j \in \{1, \ldots, J\}$. Otherwise, we call the tokeniser **sat-noncompliant**. Note that even though this section is about bottom-up tokenisers, this definition is still about direct tokenisers, meaning it contains tokens instead of merges.

We adapt the definition of 101-strings to include also all character-strings of the form $110^{+}1$ and $10^{+}11$. Considering the datasets output by Reduction 1, we know that there are no 101-strings in dataset $\mathcal{D}_1$. Further, we know that each unique 101-string appears in datasets $\mathcal{D}_2$, $\mathcal{D}_3$ and $\mathcal{D}_4$ exactly $2f'$, $2f''$ and $2f'''$ times, respectively, and exactly 3 times in $\mathcal{D}_5$ (this is due to us working with the three occurrences variant of MAX2SAT and to the fact that $\mathbf{x}_j^{\mathtt{T}} = 0^{2j-1}$ and $\mathbf{x}_j^{\mathtt{F}} = 0^{2j}$). We now prove our lemma.

Lemma 6. *If the* B-2-TOK *instance output by Reduction 2 is satisfiable, the* 3-OCC-MAX2SAT *instance which generated it is as well. Formally:* $\mathrm{Tok}_{\uparrow}^{2}(\mathrm{R2}(\mathcal{X}, \mathcal{C}, \gamma)) \implies \mathrm{3OM2S}(\mathcal{X}, \mathcal{C}, \gamma)$.

Proof. Assume this $(\mathcal{D}, K, \delta)$ instance of B-2-TOK—where $(\mathcal{D}, K, \delta) = \mathrm{R2}(\mathcal{X}, \mathcal{C}, \gamma)$—is satisfiable, i.e., that $\mathrm{Tok}_{\uparrow}^{2}(\mathrm{R2}(\mathcal{X}, \mathcal{C}, \gamma))$ evaluates to true. We must prove that, in this case, $\mathrm{3OM2S}(\mathcal{X}, \mathcal{C}, \gamma)$ also evaluates to true. Now, let $\mathbf{m}_{\mathtt{opt}}$ be the optimal solution to this $(\mathcal{D}, K, \delta)$ instance of the tokenisation problem. We know, by definition, that:

$$\mathrm{Tok}_{\uparrow}^{2}(\mathrm{R2}(\mathcal{X}, \mathcal{C}, \gamma)) \iff \left(\mathfrak{G}_{\ell}(\mathtt{tok}_{\uparrow}[\mathbf{m}_{\mathtt{opt}}], \mathcal{D}) \leq \delta\right) \tag{50}$$

We can thus prove this lemma by showing the following implication:

$$\left(\mathfrak{G}_{\ell}(\mathtt{tok}_{\uparrow}[\mathbf{m}_{\mathtt{opt}}], \mathcal{D}) \leq \delta\right) \implies \mathrm{3OM2S}(\mathcal{X}, \mathcal{C}, \gamma) \tag{51}$$

When comparing two bottom-up tokenisers, things quickly get messy because one has to not only consider the merges, but also their order. For this reason, we show that direct sat-compliant tokenisers can be transformed into bottom-up tokenisers without loss of compression quality. Thus, for the sat-compliant tokeniser, we can consider the direct tokeniser instead. The key idea for this to work is that all target strings are hit via a sequence of merges such that each intermediate merge also hits a target (which has high multiplicity), such that this target must also be included as a token by a direct tokeniser. We also compare to non-compliant direct tokenisers, which are by definition as least as strong as bottom-up tokenisers, thus we can make upper bounds on their performance.

We will now prove this lemma in six steps:

1. we prove that $\mathcal{S}_{\mathtt{opt}}$ must include all tokens of the form $11, \mathbf{x}_j^{\mathtt{T}}, \mathbf{x}_j^{\mathtt{F}}, 1\mathbf{x}_j^{\mathtt{T}}, \mathbf{x}_j^{\mathtt{T}}1, 1\mathbf{x}_j^{\mathtt{F}}, \mathbf{x}_j^{\mathtt{F}}1, \mathbf{x}_j^{\mathtt{T}}11, 11\mathbf{x}_j^{\mathtt{F}}$;
2. we prove that $\mathcal{S}_{\mathtt{opt}}$ must, in addition to the tokens above, only include tokens of the form $1\mathbf{x}_j^{\mathtt{T}}1, 1\mathbf{x}_j^{\mathtt{F}}1, 1\mathbf{x}_j^{\mathtt{T}}11, 11\mathbf{x}_j^{\mathtt{F}}1$;
3. we prove that $\mathcal{S}_{\mathtt{opt}}$ may only include, for each j, either token $1\mathbf{x}_j^{\mathtt{T}}1$ or $1\mathbf{x}_j^{\mathtt{F}}1$, and either token $11\mathbf{x}_j^{\mathtt{T}}1, 11\mathbf{x}_j^{\mathtt{T}}1$ or $1\mathbf{x}_j^{\mathtt{F}}11$,;
4. we prove that $\mathcal{S}_{\mathtt{opt}}$ may only include, for each j, either tokens $1\mathbf{x}_j^{\mathtt{T}}1$ or $1\mathbf{x}_j^{\mathtt{F}}1, 1\mathbf{x}_j^{\mathtt{F}}11$
5. we prove that for any sat-compliant $\mathcal{S}_{\mathtt{opt}}$, there exists a merge sequence $\mathbf{m}_{\mathtt{opt}}$ with the same performance;
6. finally, we prove that if $\left(\mathfrak{G}_{\ell}(\mathtt{tok}_{\uparrow}[\mathbf{m}_{\mathtt{opt}}], \mathcal{D}) \leq \delta\right)$, we can build a variable assignment which satisfies this $(\mathcal{X}, \mathcal{C}, \gamma)$ 3-OCC-MAX2SAT instance.

Note that, together, steps one to four show that $\mathcal{S}_{\mathtt{opt}}$ must be the vocabulary of a sat-compliant tokeniser; in step five, we show that we can convert any sat-compliant $\mathcal{S}_{\mathtt{opt}}$ to a merge sequence $\mathbf{m}_{\mathtt{opt}}$ without losing compression; and in step six, we will then rely on function g (defined above) to convert this merge sequence into a satisfying assignment $\chi = g(\mathcal{S}_{\mathtt{opt}})$ to 3-OCC-MAX2SAT.

LemmaProofStep 1. (Step ①). *An optimal (direct) tokeniser must include all tokens of the form* $11, \mathtt{x}^{\mathtt{T}}_j, \mathtt{x}^{\mathtt{F}}_j, 1\mathtt{x}^{\mathtt{T}}_j, \mathtt{x}^{\mathtt{T}}_j 1, 1\mathtt{x}^{\mathtt{F}}_j, \mathtt{x}^{\mathtt{F}}_j 1, \mathtt{x}^{\mathtt{T}}_j 11, 11\mathtt{x}^{\mathtt{F}}_j, 1\mathtt{x}^{\mathtt{T}}_j 1, 1\mathtt{x}^{\mathtt{F}}_j 1, 1\mathtt{x}^{\mathtt{T}}_j 11, 11\mathtt{x}^{\mathtt{F}}_j 1$*, i.e..:*

$$\left\{11, \mathtt{x}^{\mathtt{T}}_j, \mathtt{x}^{\mathtt{F}}_j, 1\mathtt{x}^{\mathtt{T}}_j, \mathtt{x}^{\mathtt{T}}_j 1, 1\mathtt{x}^{\mathtt{F}}_j, \mathtt{x}^{\mathtt{F}}_j 1, \mathtt{x}^{\mathtt{T}}_j 11, 11\mathtt{x}^{\mathtt{F}}_j, 1\mathtt{x}^{\mathtt{T}}_j 1, 1\mathtt{x}^{\mathtt{F}}_j 1, 1\mathtt{x}^{\mathtt{T}}_j 11, 11\mathtt{x}^{\mathtt{F}}_j 1\right\}_{j=1}^{J} \subseteq \mathcal{S}_{\mathtt{opt}} \tag{52}$$

Proof. We prove this step by contradiction. Assume there exists an optimal tokeniser with vocabulary $\mathcal{S}_{\text{✗}}$ which does not include $t > 0$ of the tokens above. Now, remove t arbitrarily chosen tokens in this vocabulary which are not of the form above, and replace them with the missing tokens in this set. We denote this new tokeniser's vocabulary $\mathcal{S}_{\text{✓}}$. Note that the strings in $\mathcal{D}_1$ with these missing tokens were represented with at least 2 symbols under $\mathcal{S}_{\text{✗}}$, but with a single token under $\mathcal{S}_{\text{✓}}$, i.e.,:

$$\mathfrak{G}_\ell(\mathtt{tok}_{\diamond}[\mathcal{S}_{\text{✗}}], \mathcal{D}_1) \geq (8J + 1 + t)f, \qquad \mathfrak{G}_\ell(\mathtt{tok}_{\diamond}[\mathcal{S}_{\text{✓}}], \mathcal{D}_1) = (8J + 1)f \tag{53}$$

Further, note that under $\mathcal{S}_{\text{✓}}$, we have that strings in dataset $\mathcal{D}_2$ are compressed to at most two symbols, while strings in $\mathcal{D}_3$, $\mathcal{D}_4$ and $\mathcal{D}_5$ are compressed to at most three symbols:

$$\forall \mathtt{c} \in \mathcal{D}_2 : \mathfrak{G}_\ell(\mathtt{tok}_{\diamond}[\mathcal{S}_{\text{✓}}], \mathtt{c}) \leq 2, \quad \forall \mathtt{c} \in \mathcal{D}_3 \cup \mathcal{D}_4 \cup \mathcal{D}_5 : \mathfrak{G}_\ell(\mathtt{tok}_{\diamond}[\mathcal{S}_{\text{✓}}], \mathtt{c}) \leq 3 \tag{54}$$

To improve on this compressed length, $\mathcal{S}_{\text{✗}}$ must, thus, compress strings in $\mathcal{D}_2$ to a single symbol, or strings in $\mathcal{D}_3$, $\mathcal{D}_4$ and $\mathcal{D}_5$ to one or two symbols. As before , this can only be done if the noncompliant tokens in $\mathcal{S}_{\text{✗}}$ contain 101-strings. [10] As discussed above, however, each 101-string appears, as a prefix or suffix, at most: $2f'$ times in $\mathcal{D}_2$, $2f''$ times in $\mathcal{D}_3$, $2f'''$ times in $\mathcal{D}_4$, 3 times in $\mathcal{D}_5$. This gives us a best case scenario—in which all the strings in which a new token appears are compressed to a single symbol—where:

$$\begin{aligned}\mathfrak{G}_\ell(\mathtt{tok}_{\diamond}[\mathcal{S}_{\text{✓}}], \mathcal{D}_2 \cup \mathcal{D}_3 \cup \mathcal{D}_4 \cup \mathcal{D}_5) - \mathfrak{G}_\ell(\mathtt{tok}_{\diamond}[\mathcal{S}_{\text{✗}}], \mathcal{D}_2 \cup \mathcal{D}_3 \cup \mathcal{D}_4 \cup \mathcal{D}_5) \\ \leq t(2f' + 2(2f'' + 2f''' + 3))\end{aligned} \tag{55a}$$

As the difference in Eq. (53) is of at least tf tokens, we put these together:

$$\mathfrak{G}_\ell(\mathtt{tok}_{\diamond}[\mathcal{S}_{\text{✓}}], \mathcal{D}) - \mathfrak{G}_\ell(\mathtt{tok}_{\diamond}[\mathcal{S}_{\text{✗}}], \mathcal{D}) \leq t(2f' + 2(2f'' + 2f''' + 3)) - tf \tag{56}$$

As $f > 2(2f' + 2f'' + 2f''' + 3)$, this difference is smaller than zero, implying that $\mathcal{S}_{\text{✓}}$ improves on $\mathcal{S}_{\text{✗}}$. This shows a contradiction, which completes our proof. □

LemmaProofStep 2. (Step ②). *An optimal tokeniser must include all tokens of the form* $11, \mathtt{x}^{\mathtt{T}}_j, \mathtt{x}^{\mathtt{F}}_j, 1\mathtt{x}^{\mathtt{T}}_j, \mathtt{x}^{\mathtt{T}}_j 1, 1\mathtt{x}^{\mathtt{F}}_j, \mathtt{x}^{\mathtt{F}}_j 1, \mathtt{x}^{\mathtt{T}}_j 11, 11\mathtt{x}^{\mathtt{F}}_j$*, and further only tokens of the form* $1\mathtt{x}^{\mathtt{T}}_j 1, 1\mathtt{x}^{\mathtt{F}}_j 1, 1\mathtt{x}^{\mathtt{T}}_j 11, 11\mathtt{x}^{\mathtt{F}}_j 1$*, i.e.,:*

$$\left\{11, \mathtt{x}^{\mathtt{T}}_j, \mathtt{x}^{\mathtt{F}}_j, 1\mathtt{x}^{\mathtt{T}}_j, \mathtt{x}^{\mathtt{T}}_j 1, 1\mathtt{x}^{\mathtt{F}}_j, \mathtt{x}^{\mathtt{F}}_j 1, \mathtt{x}^{\mathtt{T}}_j 11, 11\mathtt{x}^{\mathtt{F}}_j\right\}_{j=1}^{J} \subseteq \mathcal{S}_{\mathtt{opt}} \quad \text{and} \tag{57a}$$

$$\mathcal{S}_{\mathtt{opt}} \subset \left\{11, \mathtt{x}^{\mathtt{T}}_j, \mathtt{x}^{\mathtt{F}}_j, 1\mathtt{x}^{\mathtt{T}}_j, \mathtt{x}^{\mathtt{T}}_j 1, 1\mathtt{x}^{\mathtt{F}}_j, \mathtt{x}^{\mathtt{F}}_j 1, \mathtt{x}^{\mathtt{T}}_j 11, 11\mathtt{x}^{\mathtt{F}}_j, 1\mathtt{x}^{\mathtt{T}}_j 1, 1\mathtt{x}^{\mathtt{F}}_j 1, 1\mathtt{x}^{\mathtt{T}}_j 11, 11\mathtt{x}^{\mathtt{F}}_j 1\right\}_{j=1}^{J} \tag{57b}$$

Proof. As before, we prove this step by contradiction. Given step ① above, we know an optimal tokeniser includes all tokens $11, \mathtt{x}^{\mathtt{T}}_j, \mathtt{x}^{\mathtt{F}}_j, 1\mathtt{x}^{\mathtt{T}}_j, \mathtt{x}^{\mathtt{T}}_j 1, 1\mathtt{x}^{\mathtt{F}}_j, \mathtt{x}^{\mathtt{F}}_j 1, \mathtt{x}^{\mathtt{T}}_j 11, 11\mathtt{x}^{\mathtt{F}}_j$. Now, assume there exists an optimal tokeniser with vocabulary $\mathcal{S}_{\text{✗}}$ with $t > 0$ tokens which are not of the form $11, \mathtt{x}^{\mathtt{T}}_j, \mathtt{x}^{\mathtt{F}}_j, 1\mathtt{x}^{\mathtt{T}}_j, \mathtt{x}^{\mathtt{T}}_j 1, 1\mathtt{x}^{\mathtt{F}}_j, \mathtt{x}^{\mathtt{F}}_j 1, \mathtt{x}^{\mathtt{T}}_j 11, 11\mathtt{x}^{\mathtt{F}}_j$ or $1\mathtt{x}^{\mathtt{T}}_j 1, 1\mathtt{x}^{\mathtt{F}}_j 1, 1\mathtt{x}^{\mathtt{T}}_j 11, 11\mathtt{x}^{\mathtt{F}}_j 1$; we will call these tokens non-compliant here. Choose an arbitrary set of t unused compliant tokens—i.e., with form $1\mathtt{x}^{\mathtt{T}}_j 1, 1\mathtt{x}^{\mathtt{F}}_j 1, 1\mathtt{x}^{\mathtt{T}}_j 11, 11\mathtt{x}^{\mathtt{F}}_j 1$—to replace the non-compliant tokens with, forming a new tokeniser's vocabulary $\mathcal{S}_{\text{✓}}$. Both these vocabularies compress strings in $\mathcal{D}_1$ equally:

$$\mathfrak{G}_\ell(\mathtt{tok}_{\diamond}[\mathcal{S}_{\text{✗}}], \mathcal{D}_1) = (8J + 1)f, \qquad \mathfrak{G}_\ell(\mathtt{tok}_{\diamond}[\mathcal{S}_{\text{✓}}], \mathcal{D}_1) = (8J + 1)f \tag{58}$$

For strings in $\mathcal{D}_2$: if the entire string is in the vocabulary, it is encoded as a single token; else, it is represented with two symbols. Under $\mathcal{S}_{\text{✓}}$, there are $2J$ tokens covering strings in $\mathcal{D}_2$. Under $\mathcal{S}_{\text{✗}}$, there are only $(2J - t)$ tokens covering strings in $\mathcal{D}_2$. This implies:

$$\mathfrak{G}_\ell(\mathtt{tok}_{\diamond}[\mathcal{S}_{\text{✗}}], \mathcal{D}_2) = (6J + t)f', \qquad \mathfrak{G}_\ell(\mathtt{tok}_{\diamond}[\mathcal{S}_{\text{✓}}], \mathcal{D}_2) = 6Jf' \tag{59}$$

[10] This follows the same argument as in the proof of Lemma 2. Note that the extension of 101-strings to include strings of the form 110^+1 and 10^+11 does not break the argument, as the substring has to appear as a prefix or suffix to yield a saving. Thus, even though the strings of form 10^+1 are included in the new strings, we do not have to count those occurrences.

For strings in $\mathcal{D}_3$, $\mathcal{D}_4$ and $\mathcal{D}_5$, a similar argument to the previous step applies: (i) only tokens containing 101-strings can compress these datasets; (ii) each 101-string appears, as a prefix or suffix, at most $2f'' + 2f''' + 3$ times in them; (iii) each 101-string will lead to at most two symbols being saved. As $\mathcal{S}_{\text{✗}}$ differs from $\mathcal{S}_{\text{✓}}$ in t tokens, we get that it will improve on it by at most:

$$\mathfrak{G}_\ell(\texttt{tok}_{\Phi}[\mathcal{S}_{\text{✓}}], \mathcal{D}_3 \cup \mathcal{D}_4 \cup \mathcal{D}_5) - \mathfrak{G}_\ell(\texttt{tok}_{\Phi}[\mathcal{S}_{\text{✗}}], \mathcal{D}_3 \cup \mathcal{D}_4 \cup \mathcal{D}_5) \leq 2t(2f'' + 2f''' + 3) \tag{60}$$

Summing together the compression on all datasets, we get their difference is bounded by:

$$\mathfrak{G}_\ell(\texttt{tok}_{\Phi}[\mathcal{S}_{\text{✓}}], \mathcal{D}) - \mathfrak{G}_\ell(\texttt{tok}_{\Phi}[\mathcal{S}_{\text{✗}}], \mathcal{D}) \leq 2t(2f'' + 2f''' + 3) - tf' \tag{61}$$

As $f' > 2(2f'' + 2f''' + 3)$, this difference is smaller than zero, implying that $\mathcal{S}_{\text{✓}}$ improves on $\mathcal{S}_{\text{✗}}$. This shows a contradiction, which completes our proof. □

LemmaProofStep 3. (Step ③). *An optimal tokeniser must contain all tokens of the form* $11, \mathtt{x}_j^{\mathtt{T}}, \mathtt{x}_j^{\mathtt{F}}, 1\mathtt{x}_j^{\mathtt{T}}, \mathtt{x}_j^{\mathtt{T}}1, 1\mathtt{x}_j^{\mathtt{F}}, \mathtt{x}_j^{\mathtt{F}}1, \mathtt{x}_j^{\mathtt{T}}11, 11\mathtt{x}_j^{\mathtt{F}}$ *and further only tokens of the form* $1\mathtt{x}_j^{\mathtt{T}}1, 1\mathtt{x}_j^{\mathtt{F}}1, 1\mathtt{x}_j^{\mathtt{T}}11, 11\mathtt{x}_j^{\mathtt{F}}1$*, and for each* $1 \leq j \leq J$*, it must contain exactly one of* $1\mathtt{x}_j^{\mathtt{T}}1, 1\mathtt{x}_j^{\mathtt{F}}1$*, and exactly one of* $1\mathtt{x}_j^{\mathtt{T}}11, 11\mathtt{x}_j^{\mathtt{F}}1$.

Proof. As before, we prove this step by contradiction. Given step ① above, we know an optimal tokeniser includes all tokens $11, \mathtt{x}_j^{\mathtt{T}}, \mathtt{x}_j^{\mathtt{F}}, 1\mathtt{x}_j^{\mathtt{T}}, \mathtt{x}_j^{\mathtt{T}}1, 1\mathtt{x}_j^{\mathtt{F}}, \mathtt{x}_j^{\mathtt{F}}1, \mathtt{x}_j^{\mathtt{T}}11, 11\mathtt{x}_j^{\mathtt{F}}$. Further, given step ② above, we know its other tokens all have form $1\mathtt{x}_j^{\mathtt{T}}1, 1\mathtt{x}_j^{\mathtt{F}}1, 1\mathtt{x}_j^{\mathtt{T}}11, 11\mathtt{x}_j^{\mathtt{F}}1$. Now, assume there exists an optimal tokeniser with vocabulary $\mathcal{S}_{\text{✗}}$ which includes both tokens in a pair $1\mathtt{x}_j^{\mathtt{T}}1, 1\mathtt{x}_j^{\mathtt{F}}1$ or $11\mathtt{x}_j^{\mathtt{T}}1, 1\mathtt{x}_j^{\mathtt{F}}11$ for $t > 0$ such pairs, and thus neither of those two for $t > 0$ other such pairs. Then, define $\mathcal{S}_{\text{✓}}$ as a vocabulary where the $1\mathtt{x}_j^{\mathtt{F}}1$ respectively $1\mathtt{x}_j^{\mathtt{F}}11$ token of all t doubly assigned pairs are replaced with the $1\mathtt{x}_j^{\mathtt{T}}1$ respectively $11\mathtt{x}_j^{\mathtt{T}}1$ token of all uncovered pairs.

These two tokenisers achieve the same compression on $\mathcal{D}_1$ and $\mathcal{D}_2$:

$$\mathfrak{G}_\ell(\texttt{tok}_{\Phi}[\mathcal{S}_{\text{✗}}], \mathcal{D}_1 \cup \mathcal{D}_2) = (8J+1)f + 6Jf', \qquad \mathfrak{G}_\ell(\texttt{tok}_{\Phi}[\mathcal{S}_{\text{✓}}], \mathcal{D}_1 \cup \mathcal{D}_2) = (8J+1)f + 6Jf' \tag{62}$$

The tokeniser with vocabulary $\mathcal{S}_{\text{✓}}$ will then compress each string in $\mathcal{D}_3$ to 2 symbols, while $\mathcal{S}_{\text{✗}}$ will only compress the t strings of the form $1\mathtt{x}_j^{\mathtt{T}}1\mathtt{x}_j^{\mathtt{F}}1$ or $11\mathtt{x}_j^{\mathtt{F}}1\mathtt{x}_j^{\mathtt{T}}11$ for which the pair is uncovered, to 3 symbols. This will lead to a total compression of:

$$\mathfrak{G}_\ell(\texttt{tok}_{\Phi}[\mathcal{S}_{\text{✗}}], \mathcal{D}_3) = (4J+t)f'', \qquad \mathfrak{G}_\ell(\texttt{tok}_{\Phi}[\mathcal{S}_{\text{✓}}], \mathcal{D}_3) = 4Jf'' \tag{63}$$

Finally, the t doubly assigned tokens of the form $1\mathtt{x}_j^{\mathtt{F}}1$ (which $\mathcal{S}_{\text{✓}}$ does not contain) appear at most three times in $\mathcal{D}_5$ and will lead to at most one symbol being saved, leading to a bound:

$$\mathfrak{G}_\ell(\texttt{tok}_{\Phi}[\mathcal{S}_{\text{✗}}], \mathcal{D}_5) - \mathfrak{G}_\ell(\texttt{tok}_{\Phi}[\mathcal{S}_{\text{✓}}], \mathcal{D}_5) \leq 3t \tag{64}$$

Putting these compressed lengths together, we get:

$$\mathfrak{G}_\ell(\texttt{tok}_{\Phi}[\mathcal{S}_{\text{✗}}], \mathcal{D}) - \mathfrak{G}_\ell(\texttt{tok}_{\Phi}[\mathcal{S}_{\text{✓}}], \mathcal{D}) \leq 3t - tf'' \tag{65}$$

As $f'' > f''' + 3$, this difference is smaller than zero, implying that $\mathcal{S}_{\text{✓}}$ improves on $\mathcal{S}_{\text{✗}}$. This shows a contradiction, which completes our proof. For strings in $\mathcal{D}_4$ and $\mathcal{D}_5$, a similar argument to the previous step applies: (i) only tokens containing 101-strings can compress these datasets; (ii) each 101-string appears, as a prefix or suffix, at most $2f''' + 3$ times in them; (iii) each 101-string will lead to at most two symbols being saved. As $\mathcal{S}_{\text{✗}}$ differs from $\mathcal{S}_{\text{✓}}$ in t tokens, we get that it will improve on it by at most:

$$\mathfrak{G}_\ell(\texttt{tok}_{\Phi}[\mathcal{S}_{\text{✓}}], \mathcal{D}_4 \cup \mathcal{D}_5) - \mathfrak{G}_\ell(\texttt{tok}_{\Phi}[\mathcal{S}_{\text{✗}}], \mathcal{D}_4 \cup \mathcal{D}_5) \leq 2t(2f''' + 3) \tag{66}$$

Summing together the compression on all datasets, we get their difference is bounded by:

$$\mathfrak{G}_\ell(\texttt{tok}_{\Phi}[\mathcal{S}_{\text{✓}}], \mathcal{D}) - \mathfrak{G}_\ell(\texttt{tok}_{\Phi}[\mathcal{S}_{\text{✗}}], \mathcal{D}) \leq 2t(2f''' + 3) - tf' \tag{67}$$

As $f'' > 2(2f''' + 3)$, this difference is smaller than zero, implying that $\mathcal{S}_{\text{✓}}$ improves on $\mathcal{S}_{\text{✗}}$. This shows a contradiction, which completes our proof. □

LemmaProofStep 4. (Step ④). *An optimal tokeniser must be* `sat`*-compliant: it must contain all tokens of the form* $11, \mathtt{x}_j^{\mathtt{T}}, \mathtt{x}_j^{\mathtt{F}}, 1\mathtt{x}_j^{\mathtt{T}}, \mathtt{x}_j^{\mathtt{T}}1, 1\mathtt{x}_j^{\mathtt{F}}, \mathtt{x}_j^{\mathtt{F}}1, \mathtt{x}_j^{\mathtt{T}}11, 11\mathtt{x}_j^{\mathtt{F}}$ *and further only tokens of the form* $1\mathtt{x}_j^{\mathtt{T}}1, 1\mathtt{x}_j^{\mathtt{F}}1, 1\mathtt{x}_j^{\mathtt{T}}11, 11\mathtt{x}_j^{\mathtt{F}}1$*, and for each* $1 \leq j \leq J$*, it must include either* $1\mathtt{x}_j^{\mathtt{T}}1, 1\mathtt{x}_j^{\mathtt{T}}11$ *or* $1\mathtt{x}_j^{\mathtt{F}}1, 11\mathtt{x}_j^{\mathtt{F}}1$.

Proof. As before, we prove this step by contradiction. Given step ① above, we know an optimal tokeniser includes all tokens $11, \mathtt{x}_j^{\mathtt{T}}, \mathtt{x}_j^{\mathtt{F}}, 1\mathtt{x}_j^{\mathtt{T}}, \mathtt{x}_j^{\mathtt{T}}1, 1\mathtt{x}_j^{\mathtt{F}}, \mathtt{x}_j^{\mathtt{F}}1, \mathtt{x}_j^{\mathtt{T}}11, 11\mathtt{x}_j^{\mathtt{F}}$. Further, given step ② above, we know its other tokens all have form $1\mathtt{x}_j^{\mathtt{T}}1, 1\mathtt{x}_j^{\mathtt{F}}1, 1\mathtt{x}_j^{\mathtt{T}}11, 11\mathtt{x}_j^{\mathtt{F}}1$. Finally, given step ③ above, we know it contains exactly one token for each pair $1\mathtt{x}_j^{\mathtt{T}}1, 1\mathtt{x}_j^{\mathtt{F}}1$ and $11\mathtt{x}_j^{\mathtt{T}}1, 1\mathtt{x}_j^{\mathtt{F}}11$.

Now, assume there exists an optimal tokeniser with vocabulary $\mathcal{S}_{\text{✗}}$ which includes both tokens in a pair $1\mathtt{x}_j^{\mathtt{F}}1, 1\mathtt{x}_j^{\mathtt{T}}1$ or $1\mathtt{x}_j^{\mathtt{T}}1, 1\mathtt{x}_j^{\mathtt{F}}11$ for $t > 0$ such pairs, and thus neither of those two for $t > 0$ other such pairs. Then, define $\mathcal{S}_{\text{✓}}$ as a vocabulary where the $1\mathtt{x}_j^{\mathtt{F}}1$ respectively $1\mathtt{x}_j^{\mathtt{F}}11$ token of all t doubly assigned pairs are replaced with the $11\mathtt{x}_j^{\mathtt{T}}1$ respectively $1\mathtt{x}_j^{\mathtt{T}}1$ token of all uncovered pairs. These two tokenisers achieve the same compression on $\mathcal{D}_1$, $\mathcal{D}_2$ and $\mathcal{D}_3$:

$$\mathfrak{G}_\ell(\mathtt{tok}_\Diamond[\mathcal{S}_{\text{✗}}], \mathcal{D}_1 \cup \mathcal{D}_2) = (8J+1)f + 6Jf' + 4Jf'' \tag{68}$$

$$\mathfrak{G}_\ell(\mathtt{tok}_\Diamond[\mathcal{S}_{\text{✓}}], \mathcal{D}_1 \cup \mathcal{D}_2) = (8J+1)f + 6Jf' + 4Jf'' \tag{69}$$

The tokeniser with vocabulary $\mathcal{S}_{\text{✓}}$ will then compress each string in $\mathcal{D}_4$ to 2 symbols, while $\mathcal{S}_{\text{✗}}$ will only compress the t strings of the form $1\mathtt{x}_j^{\mathtt{T}}1\mathtt{x}_j^{\mathtt{F}}11$ or $11\mathtt{x}_j^{\mathtt{F}}1\mathtt{x}_j^{\mathtt{T}}1$ for which the pair is uncovered, to 3 symbols. This will lead to a total compression of:

$$\mathfrak{G}_\ell(\mathtt{tok}_\Diamond[\mathcal{S}_{\text{✗}}], \mathcal{D}_4) = (4J+t)f''', \qquad \mathfrak{G}_\ell(\mathtt{tok}_\Diamond[\mathcal{S}_{\text{✓}}], \mathcal{D}_4) = 4Jf''' \tag{70}$$

Finally, the t doubly assigned tokens of the form $1\mathtt{x}_j^{\mathtt{F}}1$ or $11\mathtt{x}_j^{\mathtt{F}}1$ (which $\mathcal{S}_{\text{✓}}$ does not contain) appear, as a prefix or suffix, at most three times in $\mathcal{D}_5$ and will lead to at most one symbol being saved, leading to a bound:

$$\mathfrak{G}_\ell(\mathtt{tok}_\Diamond[\mathcal{S}_{\text{✗}}], \mathcal{D}_5) - \mathfrak{G}_\ell(\mathtt{tok}_\Diamond[\mathcal{S}_{\text{✓}}], \mathcal{D}_5) \leq 3t \tag{71}$$

Putting these compressed lengths together, we get:

$$\mathfrak{G}_\ell(\mathtt{tok}_\Diamond[\mathcal{S}_{\text{✗}}], \mathcal{D}) - \mathfrak{G}_\ell(\mathtt{tok}_\Diamond[\mathcal{S}_{\text{✓}}], \mathcal{D}) \leq 3t - tf''' \tag{72}$$

As $f''' > 3$, this difference is smaller than zero, implying that $\mathcal{S}_{\text{✓}}$ improves on $\mathcal{S}_{\text{✗}}$. This shows a contradiction, which completes our proof.

□

LemmaProofStep 5. (Step ⑤). *Any* `sat`*-compliant direct tokeniser can be transformed into a bottom-up tokeniser without changing its performance. As any optimal direct tokeniser is* `sat`*-compliant, this implies:*

$$\mathrm{Tok}^2_\uparrow(\mathrm{R2}(\mathcal{X}, \mathcal{C}, \gamma)) \iff \mathrm{Tok}^2_\Diamond(\mathrm{R2}(\mathcal{X}, \mathcal{C}, \gamma)) \tag{73}$$

Proof. As every bottom-up tokeniser can be interpreted as a direct tokeniser with the same vocabulary size and a possibly suboptimal application of its tokens, it holds that:

$$\mathrm{Tok}^2_\uparrow(\mathrm{R2}(\mathcal{X}, \mathcal{C}, \gamma)) \implies \mathrm{Tok}^2_\Diamond(\mathrm{R2}(\mathcal{X}, \mathcal{C}, \gamma)) \tag{74}$$

This is to say, direct tokenisers always compress as least as good as bottom-up tokenisers, when allowed the same vocabulary size.

Given a `sat`-compliant direct tokeniser, we first describe how to transform it into a bottom-up tokeniser. We always include merges:

$$\mathbf{m}_1 = \langle 1, 1\rangle \circ \bigcirc_{j=1}^{\lfloor log(2J-1)\rfloor}[\langle 0^{2^i}, 0^{2^i}\rangle] \circ \bigcirc_{j=1}^{\lfloor log(2J-1)\rfloor} \bigcirc_{j'=1}^{2^j-1}[\langle 0^{2^j}, 0^{j'}\rangle] \tag{75a}$$

$$\mathbf{m}_2 = \bigcirc_{j=1}^{J}[\langle 11, \mathtt{x}_j^{\mathtt{F}}\rangle, \langle \mathtt{x}_j^{\mathtt{T}}, 11\rangle] \tag{75b}$$

$$\mathbf{m}_4 = \bigcirc_{j=1}^{J}[\langle \mathtt{x}_j^{\mathtt{F}}, 1\rangle, \langle 1, \mathtt{x}_j^{\mathtt{T}}\rangle] \tag{75c}$$

$$\mathbf{m}_6 = \bigcirc_{j=1}^{J}[\langle 1, \mathtt{x}_j^{\mathtt{F}}\rangle, \langle \mathtt{x}_j^{\mathtt{T}}, 1\rangle] \tag{75d}$$

and additionally, depending on which tokens are included in the direct tokeniser's vocabulary $\mathcal{S}$:

$$\mathbf{m}_3 = \bigcirc_{j=1}^{J}\begin{bmatrix} \langle 11, \mathtt{x}_j^{\mathtt{T}}11\rangle & \texttt{if } 11\mathtt{x}_j^{\mathtt{T}}11 \in \mathcal{S} \\ \langle 11\mathtt{x}_j^{\mathtt{F}}, 1\rangle & \texttt{else} \end{bmatrix}, \tag{76a}$$

$$\mathbf{m}_5 = \bigcirc_{j=1}^{J}\begin{bmatrix} \langle 1\mathtt{x}_j^{\mathtt{T}}, 1\rangle & \texttt{if } 1\mathtt{x}_j^{\mathtt{T}}1 \in \mathcal{S} \\ \langle 1, \mathtt{x}_j^{\mathtt{F}}1\rangle & \texttt{else} \end{bmatrix} \tag{76b}$$

Note that because the tokeniser is sat-compliant, we have

$$11\mathbf{x}_j^{\mathrm{T}}11 \in \mathcal{S} \iff 1\mathbf{x}_j^{\mathrm{T}}1 \in \mathcal{S} \tag{77}$$

It is easy to verify that the resulting bottom-up tokeniser has the same performance on datasets $\mathcal{D}_1$ to $\mathcal{D}_4$. For $\mathcal{D}_5$, Tab. 1 shows that each string which could be reduced to two symbols by the direct tokeniser is also reduced to two symbols by the bottom-up tokeniser.

Thus, we actually get:

$$\mathrm{Tok}^2_{\uparrow}(\mathrm{R2}(\mathcal{X},\mathcal{C},\gamma)) \iff \mathrm{Tok}^2_{\Updownarrow}(\mathrm{R2}(\mathcal{X},\mathcal{C},\gamma)) \tag{78}$$

□

LemmaProofStep 6. (Step ⑥). *If an optimal (direct) tokeniser achieves a compressed length of at least* $5398J + 575 + 3I - \gamma$*, the original* 3-OCC-MAX2SAT *instance is satisfiable, i.e.,:*

$$\left(\mathfrak{G}_\ell(\mathtt{tok}_{\Updownarrow}[\mathcal{S}_{\mathtt{opt}}],\mathcal{D}) \leq 5398J + 575 + 3I - \gamma\right) \implies \mathrm{3OM2S}(\mathcal{X},\mathcal{C},\gamma) \tag{79}$$

Proof. Given steps ① to ④, we know that an optimal tokeniser will be sat-compliant. We will now denote this optimal tokeniser's vocabulary as $\mathcal{S}_{\mathtt{opt}}$ and use Eq. (15) to extract a 3-OCC-MAX2SAT assignment $\mathcal{X}^\star = g(\mathcal{S}_{\mathtt{opt}})$ which corresponds to this tokeniser's vocabulary. From the previous proof steps we see that any sat-compliant tokeniser achieves the following compressed length in $\mathcal{D}_1$, $\mathcal{D}_2$, $\mathcal{D}_3$, and $\mathcal{D}_4$:

$$\begin{aligned}
\mathfrak{G}_\ell(\mathtt{tok}_{\Updownarrow}[\mathcal{S}_{\mathtt{opt}}],\mathcal{D}_1\cup\mathcal{D}_2\cup\mathcal{D}_3\cup\mathcal{D}_4) &= (8J+1)f + 6Jf' + 4Jf'' + 4Jf''' && \text{(80a)}\\
&= (8\cdot 575 + 6\cdot 115 + 4\cdot 23 + 4\cdot 4)J + 575 && \text{(80b)}\\
&= 5398J + 575 && \text{(80c)}
\end{aligned}$$

Now, note that any target string in $\mathcal{D}_5$ will be: compressed to two symbols if at least one of the tokens $1L_i^1 1$ (respectively $11L_i^1 1$, if L_i^1 is a negated literal) or $1L_i^2 11$ (respectively $1L_i^2 1$, if L_i^2 is a negated literal) exist, or compressed to three symbols if neither exists. Equivalently, a clause $L_i^1 \vee L_i^2$ in 3-OCC-MAX2SAT is: satisfied, if either L_i^1 or L_i^2 evaluates to true; not satisfied, if both evaluate to false. Given our construction of function g above, one of 3-OCC-MAX2SAT's clauses will be satisfied if and only if its corresponding string in $\mathcal{D}_4$ is compressed to two symbols. We can thus state that:

$$\left(\mathfrak{G}_\ell(\mathtt{tok}_{\Updownarrow}[\mathcal{S}_{\mathtt{opt}}],\mathcal{D}_5) = 3I - \gamma^\star\right) \iff \left(\sum_{i=1}^{I} \mathbb{1}_{\mathcal{X}^\star}\{L_i^1 \vee L_i^2\} = \gamma^\star\right) \tag{81}$$

Given the construction of δ as $5398J + 575 + 3I - \gamma$, we conclude that a sat-compliant tokeniser which compresses the full dataset to at least that size can be mapped to a 3-OCC-MAX2SAT assignment which satisfies at least γ clauses. This concludes the proof. □

Steps ⑤ and ⑥ show

$$\mathrm{Tok}^2_{\uparrow}(\mathrm{R2}(\mathcal{X},\mathcal{C},\gamma)) \iff \mathrm{Tok}^2_{\Updownarrow}(\mathrm{R2}(\mathcal{X},\mathcal{C},\gamma)) \tag{82}$$

$$\mathrm{Tok}^2_{\Updownarrow}(\mathrm{R2}(\mathcal{X},\mathcal{C},\gamma)) \implies \mathrm{3OM2S}(\mathcal{X},\mathcal{C},\gamma) \tag{83}$$

which allows us to conclude

$$\mathrm{Tok}^2_{\uparrow}(\mathrm{R2}(\mathcal{X},\mathcal{C},\gamma)) \implies \mathrm{3OM2S}(\mathcal{X},\mathcal{C},\gamma). \tag{84}$$

□

F Proof of Lemma 7

Lemma 7. *The direct binary tokenisation gap problem is* NP*-hard.*

Proof. For this proof, we again rely on a result by Berman and Karpinski (1998; 1999) that, for specific instances of 3-OCC-MAX2SAT with $I = 2016n$ clauses, it is NP-hard to distinguish whether at least $(2012 - \varepsilon)n$ or at most $(2011 + \varepsilon)n$ of those are satisfiable, for any $\varepsilon > 0$. We will denote this 3-OCC-MAX2SAT gap problem as $\text{3OM2S}(\mathcal{X}, \mathcal{C}, (\gamma^-, \gamma^+))$, with $\gamma^- = (2011 + \varepsilon)n$ and $\gamma^+ = (2012 - \varepsilon)n$. We can now prove the NP-hardness of the bottom-up binary tokenisation gap problem by reducing 3-OCC-MAX2SAT's gap problem to it. To this end, we rely on a reduction identical to $\text{R2}(\mathcal{X}, \mathcal{C}, \gamma)$, but where we define:

$$\begin{aligned} \delta^- &= 5398J + 575 + 3I - \gamma^- & \delta^+ &= 5398J + 575 + 3I - \gamma^+ \\ &= 5398J + 575 + 3I - \frac{2011 + \varepsilon}{2016} I & &= 5398J + 575 + 3I - \frac{2012 - \varepsilon}{2016} I \end{aligned} \tag{85}$$

Lemmas 5 and 6 trivially show the validity of this reduction:

$$\text{3OM2S}(\mathcal{X}, \mathcal{C}, (\gamma^-, \gamma^+)) \iff \text{Tok}^2_{\uparrow}(\mathcal{D}, K, (\delta^-, \delta^+)) \tag{86}$$

which holds since $\text{3OM2S}(\mathcal{X}, \mathcal{C}, \gamma^+) \iff \text{Tok}^2_{\uparrow}(\mathcal{D}, K, \delta^+)$ and the same for γ^- and δ^-. It is therefore NP-hard to distinguish whether a dataset can be compressed to at most $5398J + 575 + 3I - \frac{2012-\varepsilon}{2016} I$ symbols, or if at least $5398J + 575 + 3I - \frac{2011+\varepsilon}{2016} I$ symbols remain (with an allowed vocabulary size $K = 10J$). Because each variable occurs exactly three times in 3-OCC-MAX2SAT, we have that $\frac{3}{2} J = I$. We now compute the maximum achievable compression ratio:

$$\frac{\delta^-}{\delta^+} = \frac{5398J + 575 + 3I - \frac{2011+\varepsilon}{2016} I}{5398J + 575 + 3I - \frac{2012-\varepsilon}{2016} I} \tag{87a}$$

$$= \frac{10805 - \frac{6033+3\varepsilon'}{2016}}{10805 - \frac{6036-\varepsilon'}{2016}} \tag{87b}$$

$$= \frac{21776847 - 3\varepsilon'}{21776844 + 3\varepsilon'} \tag{87c}$$

$$= \frac{7258949 - \varepsilon'}{7258948 + \varepsilon'} \tag{87d}$$

Thus, bottom-up binary tokenisation cannot be approximated in polynomial time with an approximation ratio better than $\frac{7258949}{7258948} > 1.0000001$ unless P = NP. □

G Proof that Direct Unary Tokenisation is in NP

A decision problem is in the nondeterministic polynomial time class (NP) if it can be verified in polynomial time in the presence of a **certificate**: a string designed to verify that the current instance is a "yes"-instance, typically encoding an optimal solution to its search problem. In the case of D-1-TOK, this certificate could be a set of string-lengths composing the tokeniser's vocabulary $\mathcal{S}_{\mathbb{N}}$, as well as a set $\mathcal{Z} = \{\mathbf{z}_m\}_{m=1}^{M}$, where each $\mathbf{z}_m \in \mathbb{N}^{|\mathcal{S}_{\mathbb{N}}|}$ shows how many tokens of each length should be used to tokenise each string in the dataset $\mathcal{D}_{\mathbb{N}}$. Verifying this certificate then simply requires computing the sum of tokens used for each target.

Lemma 8. *The direct unary tokenisation decision problem is in* NP.

Proof. When inputs are represented as strings, this lemma follows trivially from the unbounded-alphabet case discussed in Whittington et al. (2025). We now prove this also holds when inputs are represented as string-lengths. If $|\mathcal{D}_{\mathbb{N}}| \leq K$, each $\ell \in \mathcal{D}_{\mathbb{N}}$ can be included as a token, and thus all entries in our dataset can be compressed into single token; we consequently verify the problem's satisfiability by checking if $\delta \geq |\mathcal{D}_{\mathbb{N}}|$. Assuming $K < |\mathcal{D}_{\mathbb{N}}|$—and therefore that K's value is polynomial in the input—we have that the certificate also has polynomial length, and that, in particular, the size of $\mathcal{Z}$ is bounded by $|\mathcal{D}_{\mathbb{N}}|\, K\, \log \ell_{\max}$, where $\ell_{\max}$ is the maximum string-length in $\mathcal{D}_{\mathbb{N}}$. Thus all that is left is to compute and check if $\sum_{\mathbf{z} \in \mathcal{Z}} \texttt{sum}(\mathbf{z}) \leq \delta$. □

H Proof of Lemma 9

Lemma 9. *If a* vertex-cover *instance is satisfiable, then the* D-1-TOK *instance output by Reduction 3 is also satisfiable. Formally:* $\text{VC}(\mathcal{V}, \mathcal{E}, \psi) \implies \text{Tok}^1_{\diamond}(\text{R3}(\mathcal{V}, \mathcal{E}, \psi))$.

Proof. Suppose the given instance of `vertex-cover` has a solution. Then there exists a vertex cover $\mathcal{C}^\star \subseteq \mathcal{V}$ which uses ψ vertices. Then, we can choose as tokens:

$$\mathcal{S}_{\mathbb{N}} = \{\ell_j \mid v_j \in \mathcal{V}\} \cup \{B\} \cup \{\ell'_j \mid v_j \in \mathcal{C}^\star\} \tag{88}$$

We have that every vertex string in $\mathcal{D}_1$ will be covered by a single token (either ℓ_j or B).

All cover strings in $\mathcal{D}_2$ which encode a vertex that belongs to the $\mathcal{C}^\star$ will also be covered by a single token (ℓ'_j). The remaining cover strings in $\mathcal{D}_2$ will be covered by 2 tokens, as for every target $\ell'_j = \texttt{enc}(v_j) + N^3$ there exist the tokens $\ell_j = \texttt{enc}(v_j)$ and $B = N^3$.

As $\mathcal{C}^\star$ is a vertex cover, we have that for every edge $(v_j, v_{j'}) \in \mathcal{E}$ at least one of the vertices belongs to $\mathcal{C}^\star$. It follows that for every edge string in $\mathcal{D}_3$, $\ell''_j = \texttt{enc}(v_j) + \texttt{enc}(v_{j'}) + N^3$ at least one of the tokens $\ell'_j = \texttt{enc}(v_j) + B$ or $\ell'_{j'} = \texttt{enc}(v_{j'}) + B$ belongs to $\mathcal{S}_{\mathbb{N}}$. Thus, all edge strings are covered by two tokens.

We can now count the number of tokens in each dataset: $\mathcal{D}_1$ would have at most $J+1$ symbols; $\mathcal{D}_2$ would have at most $2J - \psi$ symbols; and $\mathcal{D}_3$ would have at most $2I$ symbols. This gives us a total of at most $3J + 2I + 1 - k = \delta$ symbols, which satisfies this tokenisation instance. Thus, we have that $\text{Tok}^1_\diamond(\mathcal{D}, K, \delta) = \texttt{T}$. □

I Proof of Lemma 10

Lemma 10. *If the* `D-1-TOK` *instance output by Reduction 3 is satisfiable, then the* `vertex-cover` *instance which generated it is as well. Formally:* $\text{Tok}^1_\diamond(\text{R3}(\mathcal{V}, \mathcal{E}, \psi)) \implies \text{VC}(\mathcal{V}, \mathcal{E}, \psi)$.

Proof. Assume this $(\mathcal{D}_{\mathbb{N}}, K, \delta)$ instance of `D-1-TOK`—where $(\mathcal{D}_{\mathbb{N}}, K, \delta) = \text{R3}(\mathcal{V}, \mathcal{E}, \psi)$—is satisfiable, i.e., that $\text{Tok}^1_\diamond(\text{R3}(\mathcal{V}, \mathcal{E}, \psi))$ evaluates to true. We must prove that, in this case, $\text{VC}(\mathcal{V}, \mathcal{E}, \psi)$ also evaluates to true. Now, let $\mathcal{S}_{\mathbb{N}}$ be the optimal solution to this $(\mathcal{D}_{\mathbb{N}}, K, \delta)$ instance of the tokenisation problem. By construction, we have that $K = J + 1 + \psi$ and $\delta = 3J + 2I + 1 - \psi$. We now prove this lemma in four steps:

① We prove all strings in $\mathcal{D}_{\mathbb{N}}$ are unique;

② We prove that an optimal tokeniser must only include full character-strings in its vocabulary;

③ We prove that an optimal tokeniser will include all strings in $\mathcal{D}_1$ in its vocabulary;

④ We prove that if an optimal tokeniser achieves δ compression, than the instance of the `vertex-cover` which was reduced to it is true.

LemmaProofStep 1. (Step ①). *All strings in $\mathcal{D}_{\mathbb{N}}$ are unique.*

Proof. Now we note that, also by construction, all strings in $\mathcal{D}_{\mathbb{N}}$ are unique. These strings all have lengths:

$$\ell_j = \texttt{enc}(v_j), \quad B = N^4, \quad \ell'_j = \texttt{enc}(v_j) + B, \quad \ell''_{j,j'} = \texttt{enc}(v_j) + \texttt{enc}(v_{j'}) + B \tag{89}$$

for $1 \leq j, j' \leq J$ and with $\texttt{enc}(v_j) = j + j^2 N + j^3 N^2$. Notably, our reduction defines $N \gg J$ and it will be useful to think about these lengths in base N. Let a number $(a, b, c, d, e)_N$ denote $aN^4 + bN^3 + cN^2 + dN^1 + e$. We can write in this base:

$$(a, b, c, d, N-1)_N + (a, 0, 0, 0, 1)_N = (2a, b, c, d+1, 0)_N \tag{90}$$

We can similarly write in this base:

$$B = (1, 0, 0, 0, 0)_N, \qquad \ell'_j = (1, 0, j^3, j^2, j)_N, \tag{91a}$$

$$\ell''_{j,j'} = (1, 0, j^3 + j'^3, j^2 + j'^2, j + j')_N, \qquad \ell'_j + \ell'_{j'} = (2, 0, j^3 + j'^3, j^2 + j'^2, j + j')_N \tag{91b}$$

As is usual, two numbers are the same only if each "digit" in this base system is the same. Given this structure, it should be clear that ℓ_j and B are all unique string-lengths. Further, the string-lengths ℓ'_j are all different from one another. It is left to show that: (i) all string-lengths ℓ'_j are different

from all $\ell''_{j,j'}$; and (ii) that string-lengths $\ell''_{j,j'}$ are different among themselves. Point (i) is proven by Lemma 11, which shows that there is no set of numbers $j, j', j'' \in \mathbb{N}$ for which $j = j' + j''$ and $j^2 = j'^2 + j''^2$. Point (ii) is proven by Lemma 12, which shows that there is no set of numbers $j, j', j'', j''' \in \mathbb{N}$ for which $j + j' = j'' + j'''$ and $j^2 + j'^2 = j''^2 + j'''^2$. □

LemmaProofStep 2. (Step ②). *An optimal tokeniser must only include full character-strings in $\mathcal{D}_{\mathbb{N}}$ and compress all other strings to two symbols.*

Proof. Note that, since all strings in $\mathcal{D}_{\mathbb{N}}$ are unique, the best compression one could (hypothetically) achieve would result from compressing: K strings to a single symbol, $|\mathcal{D}_{\mathbb{N}}| - K$ to two symbols. As $|\mathcal{D}_{\mathbb{N}}| = 2J + I + 1$ strings, this (hypothetical) optimal compression would lead to:

$$\mathfrak{G}_{\ell}(\mathcal{S}_{\texttt{opt}}, \mathcal{D}_{\mathbb{N}}) = K + 2(|\mathcal{D}_{\mathbb{N}}| - K) \tag{92a}$$
$$= J + 1 + \psi + 2(2J + I + 1 - J - 1 - \psi) \tag{92b}$$
$$= 3J + 2I + 1 + \psi - \psi) = \delta \tag{92c}$$

As by assumption $\text{Tok}^1_*(\text{R3}(\mathcal{V}, \mathcal{E}, \psi))$ evaluates to true, our tokeniser must achieve this compression, and must thus only include K full strings in $\mathcal{D}_{\mathbb{N}}$. Further, it must compress all other strings to at most two symbols. □

LemmaProofStep 3. (Step ③). *An optimal tokeniser selects every vertex string in $\mathcal{D}_1$ as a token.*

Proof. Suppose that some type vertex string ℓ_j in $\mathcal{D}_1$ is not chosen as a token. Then ℓ_j must be the sum of two tokens. No tokens of cover or vertex strings (in datasets $\mathcal{D}_2$ and $\mathcal{D}_3$) can be used, since such tokens contain a summand B, which is significantly larger than ℓ_j. Hence, both summands would have to also be vertex strings. Without loss of generality, they would have values:

$$\ell_j = (1, 0, j^3, j^2, j)_N, \quad \ell_{j'} = (1, 0, j'^3, j'^2, j')_N, \quad \ell_{j''} = (1, 0, j''^3, j''^2, j'')_N \tag{93}$$

Again, by Lemma 11, it is impossible for $\ell_j = \ell_{j'} + \ell_{j''}$. This concludes the proof that all character-strings in $\mathcal{D}_1$ must be included in the vocabulary $\mathcal{S}_{\mathbb{N}}$. Further, every cover and edge string is larger than B, while all vertex strings are significantly smaller than it; B thus cannot be written as two other tokens and must thus also be part of $\mathcal{S}_{\mathbb{N}}$. □

LemmaProofStep 4. (Step ④). *If an optimal tokeniser achieves compression δ, the original* `vertex-cover` *instance is satisfiable.*

Proof. After showing all strings in $\mathcal{D}_1$ are tokens, we have ψ remaining tokens to pick. Without loss of generality, let $0 \leq s \leq \psi$ of these remaining tokens be edge strings (from $\mathcal{D}_3$) and the remaining $\psi - s$ be cover strings ((from $\mathcal{D}_2$). Select any of the selected edge tokens $\ell''_{j_1,j_2} = (1, 0, j_1^3 + j_2^3, j_1^2 + j_2^2, j_1 + j_2)_N$. For this token to be used to compressed another string, we must have that there exists a token $\ell^?_?$ which:

$$\ell_{j_3} - \ell''_{j_1,j_2} = \ell^?_? = (-1, 0, j_3^3 - j_1^3 - j_2^3, j_3^2 - j_1^2 - j_2^2, j_3 - j_1 - j_2)_N \tag{94a}$$
$$\ell'_{j_3} - \ell''_{j_1,j_2} = \ell^?_? = (0, 0, j_3^3 - j_1^3 - j_2^3, j_3^2 - j_1^2 - j_2^2, j_3 - j_1 - j_2)_N \tag{94b}$$
$$\ell''_{j_3,j_4} - \ell''_{j_1,j_2} = \ell^?_? = (0, 0, j_3^3 + j_4^3 - j_1^3 - j_2^3, j_3^2 + j_4^2 - j_1^2 - j_2^2, j_3 + j_4 - j_1 - j_2)_N \tag{94c}$$

The third case can clearly not be satisfied, as there are no negative string-lengths in our dataset. The two first ones would need to be satisfied by a vertex string ℓ_j, as these are the only strings smaller than B. However, no number with form $\ell_j = (0, 0, j^3, j^2, j)_N$ can satisfy these equalities per Lemmas 13 and 14 (respectively, for Eqs. (94b) and (94c)). In other words, this shows that edge strings cannot contribute to any other target value. We are thus left with $\psi - s$ tokens formed of cover strings. These tokens must be used, in conjunction with vertex strings, to compress all the remaining edge strings to two tokens. Note that only composing a cover and a vertex string can compress a edge string to two symbols:

$$\ell_{j_1} + \ell_{j_2} = (0, 0, j_1^3 + j_2^3, j_1^2 + j_2^2, j_1 + j_2)_N \tag{95}$$
$$\ell'_{j_1} + \ell'_{j_2} = (2, 0, j_1^3 + j_2^3, j_1^2 + j_2^2, j_1 + j_2)_N \tag{96}$$
$$\ell'_{j_1} + \ell_{j_2} = (1, 0, j_1^3 + j_2^3, j_1^2 + j_2^2, j_1 + j_2)_N \tag{97}$$

This means that, for each edge string $\ell''_{j,j'}$ not in our tokeniser, we must have a token (either ℓ'_j or $\ell'_{j'}$) which "covers" it to obtain our target compression. Consider thus, the subgraph $\mathcal{V}, \mathcal{E}'$, where

$$\mathcal{E}' = \mathcal{E} \setminus \{(v_j, v_{j'}) \in \mathcal{E} \mid \ell''_{j,j'} \notin \mathcal{S}_{\mathbb{N}}\} \tag{98}$$

We have a $\psi - s$ vertex cover for this subgraph composed of vertices $\{v_j \in \mathcal{V} \mid \ell'_j \in \mathcal{S}_{\mathbb{N}}\}$. Now, if we expand this set of $\psi - s$ vertices by picking one arbitrary vertex for each edge string $\ell''_{j,j'}$ in our vocabulary, we get a cover $\mathcal{C} = \{v_j \in \mathcal{V} \mid \ell'_j \in \mathcal{S}_{\mathbb{N}}\} \cup \{v_j \mid \ell''_{j,j'} \in \mathcal{S}_{\mathbb{N}}\}$ of size at most ψ for the original graph. Thus, it follows $\mathrm{VC}(\mathcal{V}, \mathcal{E}, \psi) = \mathtt{T}$. □

We have seen than an optimal tokeniser has to admit a certain form, and that from this form we can deduce a valid vertex cover. This concludes the proof. □

We now show the technical lemmas used in the previous proof.

Lemma 11. *For any $r \in \mathbb{N}$, there do not exist non-zero $i, j \in \mathbb{N}$ such that*

$$i + j = r, \qquad i^2 + j^2 = r^2$$

Proof. From $(i+j)^2 = i^2 + 2ij + j^2$ and the hypotheses $i + j = r$ and $i^2 + j^2 = r^2$,

$$r^2 = i^2 + j^2 + 2ij = r^2 + 2ij \;\Rightarrow\; ij = 0, \tag{99}$$

contradicting $i, j > 0$. □

Lemma 12. *There do not exist two distinct pairs $\{i, j\} \neq \{a, b\}$ of positive integers such that*

$$i + j = a + b, \qquad i^2 + j^2 = a^2 + b^2 \tag{100}$$

Proof. From $i + j = a + b$ there is an integer s with

$$a = i - s, \qquad b = j + s. \tag{101}$$

Then

$$a^2 + b^2 - (i^2 + j^2) = (i - s)^2 + (j + s)^2 - i^2 - j^2 = 2s^2 + 2s(j - i). \tag{102}$$

By the second hypothesis $a^2 + b^2 = i^2 + j^2$, so $2s^2 + 2s(j - i) = 0$, i.e.

$$s\big(s + j - i\big) = 0. \tag{103}$$

Hence either $s = 0$ or $s = i - j$. If $s = 0$, then $a = i$ and $b = j$. If $s = i - j$, then $a = i - (i - j) = j$ and $b = j + (i - j) = i$. In both cases $\{a, b\} = \{i, j\}$, contradicting distinctness. □

Lemma 13. *For any $r \in \mathbb{N}$, there do not exist non-zero $i, j, k \in \mathbb{N}$ such that*

$$i + j + k = r, \qquad i^2 + j^2 + k^2 = r^2, \qquad i^3 + j^3 + k^3 = r^3. \tag{104}$$

Proof. Using $(i + j + k)^2 = i^2 + j^2 + k^2 + 2(ij + ik + jk)$ and the first two hypotheses,

$$r^2 = r^2 + 2(ij + ik + jk) \;\Rightarrow\; ij + ik + jk = 0. \tag{105}$$

With $i, j, k > 0$, each product ij, ik, jk is positive, which shows a contradiction. Hence, no solution exists. □

Lemma 14. *Let $r, p \in \mathbb{N}$. There do not exist non-zero $i, j, k \in \mathbb{N}$ such that*

$$i + j + k = r + p, \qquad i^2 + j^2 + k^2 = r^2 + p^2, \qquad i^3 + j^3 + k^3 = r^3 + p^3. \tag{106}$$

Proof. Let $p_1 = i+j+k$, $p_2 = i^2+j^2+k^2$, $p_3 = i^3+j^3+k^3$, and $e_1 = i+j+k$, $e_2 = ij+ik+jk$, $e_3 = ijk$. From the first two equations,

$$e_1 = r + p, \qquad e_2 = \frac{p_1^2 - p_2}{2} = \frac{(r+p)^2 - (r^2 + p^2)}{2} = rp. \tag{107}$$

Newton's identity for three variables gives

$$p_3 = e_1 p_2 - e_2 p_1 + 3e_3. \tag{108}$$

Substituting $p_1 = r + p$, $p_2 = r^2 + p^2$, $e_2 = rp$ yields

$$p_3 = (r+p)(r^2+p^2) - rp(r+p) + 3e_3 = (r+p)\big(r^2 + p^2 - rp\big) + 3e_3 = r^3 + p^3 + 3e_3.$$

By the third equality $p_3 = r^3 + p^3$, hence $3e_3 = 0$ and so $e_3 = ijk = 0$, contradicting $i, j, k > 0$. □

J Definition of the Addition Chain Problem

An addition chain is a sequence of integers that provides an efficient way to "build" a target set of numbers starting from 1.

Definition 4. *Let $T = \{t_1, t_2, \ldots, t_J\}$ be a finite set of positive integers. An **addition chain** for T is a sequence of integers $S = \langle s_0, s_1, \ldots, s_L \rangle$ with the following properties:*

1. *The sequence starts with $s_0 = 1$.*
2. *Every subsequent element s_i is the sum of two preceding elements:*
$$s_i = s_j + s_k, \quad \text{for some } k \leq j < i.$$
3. *The sequence contains all targets: for every $t_\ell \in T$, there is some $s_k \in S$ such that $t_\ell = s_k$.*

The length *of the chain is L.*

Definition 5. *Given a set of positive integers T, the **addition chain optimisation problem** is to find an addition chain for T whose length r is minimised.*

Definition 6. *Given a set of positive integers T and a limit L, the **addition chain decision problem** (`add-chain`) is to find an addition chain for T such that its length r is at most L.*

We denote by $\text{AddChain}(T, L)$ the corresponding decision predicate, which returns `T` if such a chain exists, and `F` otherwise.

K Proof of Theorem 6

Theorem 6. *The unary optimal-pair-encoding decision problem is weakly* NP*-complete.*

Proof. We write $\text{OPE-1-TOK}(\mathcal{D}, K, \delta)$ for a function which returns `T` if its input is achievable under a `OPE-1-TOK` decision problem, and `F` otherwise. To prove weak NP-completeness, we must show the problem is in NP and that it is weakly NP-hard. Inclusion in NP is established by (Kozma and Voderholzer, 2024). We prove weak NP-hardness via a reduction from the `add-chain` decision problem. First, we define this reduction.

Reduction 4. *Given an instance of* `add-chain` *consisting of a target set $T = \{t_1, \ldots, t_J\}$ and a length limit L, we construct an instance of the* `OPE-1-TOK` *problem with the following parameters. The dataset $\mathcal{D}$ is the set of unary strings corresponding to the targets: $\mathcal{D} = \{a^{t_1}, a^{t_2}, \ldots, a^{t_J}\}$. The merge budget is set to the addition chain length limit: $K = L$. The token count threshold is set to the number of targets $\delta = J$.*

Note that setting the threshold δ equal to the number of strings in the dataset implies that a valid solution must represent every string as a single token. The proof proceeds in two parts, showing both directions of the equivalence

$$\text{AddChain}(T, L) \iff \text{OPE-1-TOK}(\mathcal{D}, K, \delta) \tag{109}$$

We first show that a solution to the `add-chain` problem implies a solution to the `OPE-1-TOK` problem

$$\text{AddChain}(T, L) \implies \text{OPE-1-TOK}(\mathcal{D}, K, \delta) \tag{110}$$

Assume there exists a valid addition chain $S = \langle s_0, s_1, \ldots, s_L \rangle$ of length $r \leq L$ for the target set T. By definition, for each element $s_i \in S$ (where $i \geq 1$), there exist indices $j, k < i$ such that $s_i = s_j + s_k$.

We construct a merge sequence $\mathbf{m} = \langle m_1, \ldots, m_L \rangle$ of length r where each merge is defined as $m_i = \langle a^{s_j}, a^{s_k} \rangle$, corresponding to the predecessors of s_i in the addition chain.

The length of this merge sequence $\mathbf{m}$ is L, satisfying the merge budget $K = L$. By the iterative definition of the merge-extracted vocabulary, the resulting vocabulary $\mathcal{S}_\mathbf{m}$ will contain a token a^{s_i} for every element s_i in the addition chain S.

Since the addition chain S contains all targets $t_\ell \in T$, the vocabulary $\mathcal{S}_\mathbf{m}$ is guaranteed to contain a single token for each target string $a^{t_\ell} \in \mathcal{D}$. Consequently, the direct encoding function $\texttt{tok}_{\Rightarrow}[\mathcal{S}_\mathbf{m}]$ can represent each string $\mathbf{c} \in \mathcal{D}$ with exactly one token. The total token count is therefore:

$$\sum_{\mathbf{c}\in\mathcal{D}} |\texttt{tok}_{\Rightarrow}[\mathcal{S}_\mathbf{m}](\mathbf{c})| = \sum_{\mathbf{c}\in\mathcal{D}} 1 = |\mathcal{D}| \tag{111}$$

By the construction in our reduction, $|\mathcal{D}| = \delta$. The condition is met, thus proving the implication.

Next, we show that a solution to the `OPE-1-TOK` problem implies a solution to the `add-chain` problem

$$\text{OPE-1-TOK}(\mathcal{D}, K, \delta) \implies \text{AddChain}(T, L) \tag{112}$$

Assume there exists a merge sequence $\mathbf{m}$ of length $K = L$ that satisfies the `OPE-1-TOK` decision problem. The condition is:

$$\sum_{\mathbf{c}\in\mathcal{D}} |\texttt{tok}_{\Rightarrow}[\mathcal{S}_\mathbf{m}](\mathbf{c})| \leq \delta \tag{113}$$

Since the tokenization of any string must contain at least one token, the sum is lower-bounded by $|\mathcal{D}|$. By the reduction's construction, we set $\delta = |\mathcal{D}|$. Therefore, the inequality must hold with equality, which is only possible if every string is tokenized into exactly one token:

$$|\texttt{tok}_{\Rightarrow}[\mathcal{S}_\mathbf{m}](\mathbf{c})| = 1 \quad \text{for all } \mathbf{c} \in \mathcal{D}. \tag{114}$$

This implies that for every target $t_\ell \in T$, the corresponding string a^{t_ℓ} must exist as a single token in the merge-extracted vocabulary $\mathcal{S}_\mathbf{m}$.

The construction of $\mathcal{S}_\mathbf{m}$ itself certifies an addition chain for the lengths of its tokens. Let $\mathcal{L}_i$ be the set of lengths of the tokens in the vocabulary after i merges. Then $\mathcal{L}_0 = \{1\}$, and $\mathcal{L}_i = \mathcal{L}_{i-1} \cup \{\ell_j + \ell_k \mid m_i = \langle a^{\ell_j}, a^{\ell_k} \rangle\}$. The set of all token lengths in $\mathcal{S}_\mathbf{m}$ is $\mathcal{L}_K$. Since this set contains all targets $t_\ell \in T$ and was constructed from $K = L$ merges (additions), it demonstrates the existence of an addition chain for T of length at most L. This proves the implication. □